NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

IDENTIFICATION OF LOW-LATENCY OBFUSCATED TRAFFIC USING MULTI-ATTRIBUTE ANALYSIS

by

Kevin R. Dougherty

March 2017

Thesis Advisor: Shelley Gallup
Co-Advisor: Thomas Anderson

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704–0188*
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington, DC 20503.			
1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** March 2017	**3. REPORT TYPE AND DATES COVERED** Master's thesis	
4. TITLE AND SUBTITLE IDENTIFICATION OF LOW-LATENCY OBFUSCATED TRAFFIC USING MULTI-ATTRIBUTE ANALYSIS		**5. FUNDING NUMBERS**	
6. AUTHOR(S) Kevin R. Dougherty			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000		**8. PERFORMING ORGANIZATION REPORT NUMBER**	
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A		**10. SPONSORING/ MONITORING AGENCY REPORT NUMBER**	
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB number ____N/A____.			
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release. Distribution is unlimited.		**12b. DISTRIBUTION CODE**	

13. ABSTRACT (maximum 200 words)

There is no process or system capable of detecting obfuscated network traffic on Department of Defense (DOD) networks, and the quantity of obfuscated traffic on DOD networks is unknown. The presence of this traffic on a DOD network creates significant risk from both insider-threat and network-defense perspectives. This study used quantitative correlation and simple network-traffic analysis to identify common characteristics, relationships, and sources of obfuscated traffic. Each characteristic was evaluated individually for its ability to detect obfuscated traffic and in combination in a set of Naive Bayes multi-attribute prediction models. The best performing evaluations used multi-attribute analysis and proved capable of detecting approximately 80 percent of obfuscated traffic in a mixed dataset. By applying the methods and observations of this study, the threat to DOD networks from obfuscation technologies can be greatly reduced.

14. SUBJECT TERMS Tor, onion routing, obfuscation, network traffic analysis, multi-attribute analysis			**15. NUMBER OF PAGES** 109
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UU

NSN 7540–01-280-5500 Standard Form 298 (Rev. 2–89)
Prescribed by ANSI Std. 239–18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release. Distribution is unlimited.

IDENTIFICATION OF LOW-LATENCY OBFUSCATED TRAFFIC USING MULTI-ATTRIBUTE ANALYSIS

Kevin R. Dougherty
Lieutenant, United States Navy
B.G.S., Fort Hayes State University, 2009

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN CYBER SYSTEMS AND OPERATIONS

from the

NAVAL POSTGRADUATE SCHOOL
March 2017

Approved by: Shelley Gallup, Ph.D.
Thesis Advisor

Thomas Anderson, Ph.D.
Co-Advisor

Cynthia Irvine, Ph.D.
Chair, Cyber Academic Group

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

There is no process or system capable of detecting obfuscated network traffic on Department of Defense (DOD) networks, and the quantity of obfuscated traffic on DOD networks is unknown. The presence of this traffic on a DOD network creates significant risk from both insider-threat and network-defense perspectives. This study used quantitative correlation and simple network-traffic analysis to identify common characteristics, relationships, and sources of obfuscated traffic. Each characteristic was evaluated individually for its ability to detect obfuscated traffic and in combination in a set of Naive Bayes multi-attribute prediction models. The best performing evaluations used multi-attribute analysis and proved capable of detecting approximately 80 percent of obfuscated traffic in a mixed dataset. By applying the methods and observations of this study, the threat to DOD networks from obfuscation technologies can be greatly reduced.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

BBNM	Behavior-Based Network Management
DefecTor	DNS-enhanced fingerprinting and egress correlation on Tor
DH	Diffie-Hellmann
DOD	Department of Defense
ERSPAN	encapsulated remote switched port analyzer
FNR	false negative rate
FPR	false positive rate
FQDN	fully qualified domain name
FTP	file transfer protocol
GUI	graphical user interface
GUID	globally unique identifier
HTTP	HyperText Transfer protocol
HTTPS	HyperText Transfer Protocol Secure
IIS	Internet Information System
IP	Internet protocol
KCC	Key Cyber Concept
NIDS	network intrusion detection system
NPS	Naval Postgraduate School
NRL	Naval Research Lab
OR	onion routing
OTV	obfuscated threshold value
P2P	peer-to-peer
PCAP-NG	packet capture–next generation
PKI	public key infrastructure
RDP	remote desktop protocol
RSA	Rivest, Shamir and Adleman (encryption technique)
RSPAN	remote switched port analyzer
RTT	round trip time
SOCKS	socket secure (protocol)
SPAN	switched port analyzer

TCP	transmission control protocol
TLD	top-level domain
TLS	transport layer security
TOR	The Onion Routing
TPR	true positive rate
VRDM	Vector Relational Database Model
WOV	weighted obfuscation value
WWW	world wide web

ACKNOWLEDGMENTS

I would like to thank my beautiful wife, Ashley, for supporting me throughout my entire career. As a military spouse, entrepreneur, and mother to our beautiful daughter, you truly inspire me every day. Avery, your mother is the strongest and most resilient woman I know, and every day I strive to raise you to be like her.

To my thesis advisors, I appreciate your insight and "bumpers" along the way that kept my research focused, relevant, and most importantly, on-track. The knowledge and wisdom I have gained will follow me through the rest of my career, and I thank you for enriching that experience. I would also like to thank Noel Yucuis from the Dudley Knox Library's Graduate Writing Center. You helped hone my ability to write clearly and concisely; I truly thank you for all the time you spent molding my thesis and writing into what it is now.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

Obfuscation of network traffic is a process designed to circumvent censorship and specific user identification on the Internet [1]. Most obfuscation techniques in use today are based on the research by well-known cryptographer David Chaum, who created the first high-latency anonymous routing model [2]. In Chaum's model, commonly referred to as MixNet, a series of proxies were used to ensure that nodes along the transmission path could not identify the sender or recipient of the information [3], [4]. Since 1996, popular low-latency obfuscation providers, such as Tor, have attempted to hide the source of network traffic by routing traffic through a network of voluntary anonymous nodes before delivering the traffic to its requested destination [5]. Extensive research has been conducted to identify innovative methods for detecting obfuscated traffic, and researchers have had reasonable success using traffic analysis [1], [2], [4], [6]–[10] and packet entropy [1] as well as heuristic [4], semantic [1], [6], [11], deep packet inspection (DPI) [12], and machine learning–based techniques [1], [4].

Research by Wang et al. [1] combined these approaches and detected obfuscated traffic with a false positive rate (FPR) ranging from 0.002–0.00003 and a true positive rate (TPR) of 0.98–1.0. However, as new methods are discovered to identify obfuscated traffic, changes are made to each obfuscation technique in an attempt to circumvent detection. To counter this, a dynamic model incorporating a combination of identification techniques, blacklisting, and evaluation can be specified in a multi-attribute evaluation model. Using a multi-attribute approach, obfuscation indicators from disparate databases can be aggregated and evaluated to determine the likeliness of obfuscation.

A. RESEARCH MOTIVATION

Presently, there is no system or process deployed on the Department of Defense's (DOD) networks capable of detecting obfuscated network traffic, and it is not known what quantity of traffic on DOD networks is obfuscated. This poses a significant security risk from both insider threat and network defense perspectives. However, by using advanced traffic analysis [1] and blacklisting [13], it is possible to identify many popular

types of obfuscation. Additionally, by aggregating each indication, or signature, into a multi-attribute detection model, it may be possible to identify obfuscated traffic on DOD networks both forensically and in real time [6]. Once indicators are identified, system administrators can establish policies for response to obfuscated traffic; this technique could reduce network activity to only that which is attributable and provides means to hold persons accountable.

B. PURPOSE AND SCOPE

The purpose of this research is to identify obfuscation indicators, or Key Cyber Concepts (KCC), from the current body of research that can be used on a test network to evaluate whether low-latency Transmission Control Protocol/Internet Protocol (TCP/IP), specifically HyperText Transfer Protocol Secure (HTTPS) traffic, is employing obfuscation techniques. The KCCs will be statistically evaluated using R-script, then a multi-attribute analysis model will attempt to detect whether traffic is obfuscated. This thesis attempts to answer the following research questions:

1. Can low-latency obfuscated network traffic be identified in real time?
2. What IP traffic indications can be used to identify obfuscated low-latency network traffic?
3. Can multiple indications be incorporated into a multi-attribute analysis model to accurately identify obfuscated traffic?
4. Can a multi-attribute analysis model be used in a tool to provide a real-time processing capability to analyze obfuscated traffic data for automated response?

C. THESIS ORGANIZATION

The remainder of this thesis is organized as follows:

Chapter II focuses the background and previous work to define network traffic obfuscation with an emphasis on The Onion Routing (Tor). An emphasis will be placed on Tor's history, purpose, common uses, vulnerabilities, and identifiable characteristics.

Chapter III explains the methodology to be used, data generation, construction of the virtual lab, KCCs, as well as the multi-attribute analysis model.

Chapter IV examines the captured data for each KCC, their inter-attribute correlation, and the effectiveness of both single attribute and multi-attribute decision models.

Chapter V reviews the final results of the research and provides suggestions into its real-world application on DOD networks. Recommendations for future research opportunities will be also be presented.

THIS PAGE INTENTIONALLY LEFT BLANK

II. BACKGROUND AND PREVIOUS WORK

A. BACKGROUND

The World Wide Web has become the modern venue for expression and censorship due to its widespread assimilation into everyday life. As a result, multiple technologies have been developed both to facilitate a user's access to the Web, and to prevent it. The tools created to facilitate uncensored or anonymous browsing fall into the category of obfuscation technologies. The current body of research into obfuscation technologies has focused both on how to better obfuscate a user's activity and on how to detect or "de-anonymize" it. Research from [4] and [14] contend that the most widely used obfuscation technology today is Tor. As a result, a significant amount of research has been focused on Tor's design, operating characteristics, and vulnerabilities.

Early in the Naval Research Lab's (NRL) development of what is now known as Tor, Goldschlag, Syverson, and Reed contended it was not their intention to create an anonymous routing system. Instead, as stated by Syverson, the intent was to "separate identification from routing" [5], [15]. In doing so, Tor was created to connect to a series of nodes to obfuscate the association between user and activity [15]. A large factor in Tor's ability to obscure activity comes from its release as open-source software. Best put by Syverson, "A basic dilemma arises from the difficulty of trying to be anonymous by yourself" [5]. To that end, the earliest version of Tor was released to the public in 1996 with the intention that its wide-spread adoption would add an additional layer of security to the anonymous routing network by increasing its anonymity set [16].[1]

1. Early Onion Routing

The first generation of onion routing (OR) sought to lower the vulnerability traffic analysis created for network traffic and subsequent user association. The term *onion*, as defined by Goldschlag et al., refers to the data structure composed of the layers of encryption constructed around the payload between initiator and responder. Specifically,

[1] Dingledine, et al. define the anonymity set as the group of users employing a single point for traffic flow for the purpose of obfuscating their browsing activity [16].

the initiator's proxy node created the path, or virtual circuit, to the responder and encapsulated each encryption layer like the layers of an onion. To complete the creation of the virtual circuit, the initiator's proxy node sent the onion along the route. By design, each node "knew" only the identity of adjacent nodes, and the responder knew only the identity of the last node in the path. As a result, the initiator and responder were protected from simple traffic-analysis correlation [15].

According to [5] and [15], to create the onion, the initiator's proxy identified the entire virtual circuit and formed the onion in reverse order with the responder's *layer* at the core. Specifically, the encryption for the responder's proxy was constructed around the payload, and then each preceding node's encryption layer was wrapped back to the first node. The encryption layer included the required encryption keysets and expiration time, enabling bidirectional communication and replay attack protection.[2] As a result, each node knew the preceding node and where it should forward the onion to but nothing else. The only exception was the last node, which knew the responder's identity [15]. Syverson acknowledged the inherent vulnerabilities in sending the symmetric key within the onion in the first two generations of Tor, but the NRL incorporated a symmetric key instead of Diffie-Hellman (DH)–based circuit building to achieve a greater computational efficiency over forward secrecy of Tor [5].[3]

Figure 1 depicts a forward encryption onion from a notional initiator proxy that flows through nodes X and Y to node Z [15].

[2] A replay attack occurs when an attacker intercepts a message stream between sender and receiver and then resends, or replays, a duplicate message to either party [17].

[3] Forward secrecy provides protection against unauthorized viewing or collection. To ensure forward secrecy, DH key exchange and ephemeral keys are used to protect the confidentiality of a message even if the DH-based encryption is compromised. In the event of a compromise, the message would remain protected since the ephemeral key used would have expired when the original session was terminated [18].

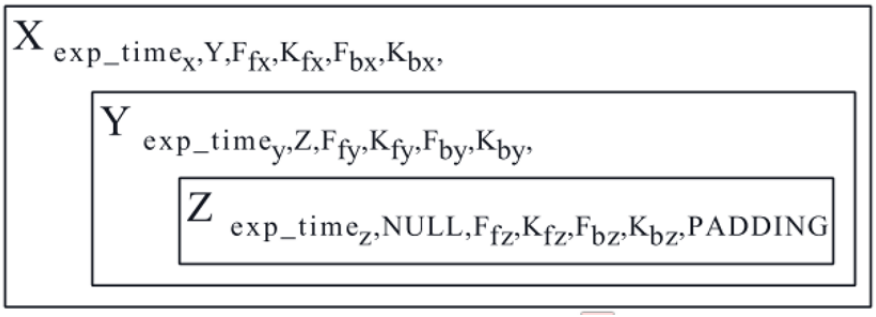


Figure 1. Complete forward onion. Source: [15].

As described by [15], each layer was constructed as follows:

$\{exp\_time,\ next\_hop,\ F_f,\ K_f,\ F_b,\ K_b,\ payload\}PK_x$, where:

- exp_time was the time the onion expired, where expiration protects against replay attacks;
- next_hop was the next hop in the onion route;
- Symmetric key $(F_f,\ K_f)$ applied to data flowing forward;
- Symmetric key $(F_b,\ K_b)$ applied to data flowing backward; and
- The padding concatenated to the end of the payload was constructed from random sized bit strings from each layer and appended before forwarding. This was done to minimize the possibility of a compromised node systematically stripping off the layers of encrypted routing information to find out where it is in the routing chain. Padding was added to all onions to fix the size and hide the length of the chain from initiator to responder.

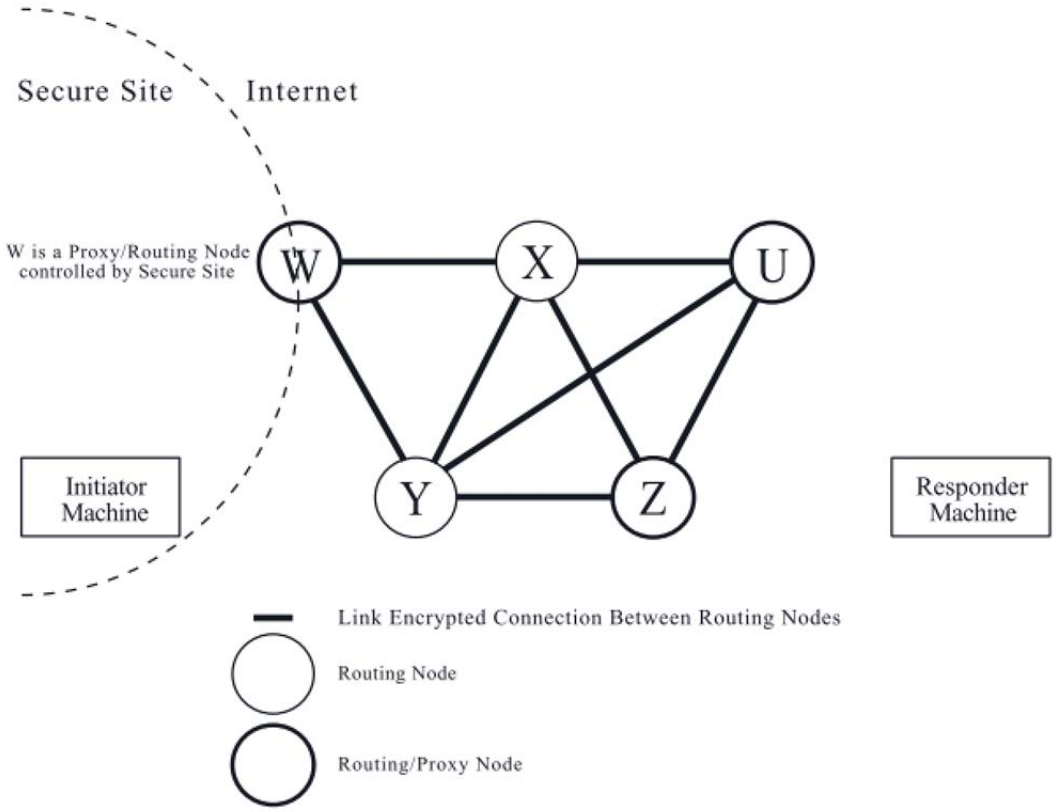
2. Generational Evolution

In total, there have been three generations of OR designed by NRL with the third being the iteration in use today. Although the functionality has remained largely unchanged through each generation, there were distinct characteristics and configurations used to disassociate users from activity in each one [5].

(1) First generation OR

- Fixed five-node (instead of three-node) routing circuits were used because they allowed communication between two enclave firewalls with a fixed

three-hop route between them to disassociate the origin and destination of the traffic. If only three hops had been used in this configuration, the middle node would have known both origin and destination of the traffic [5]. Figure 2 depicts the general topology of a first-generation onion route.



The proxy node W is the source of the communication, nodes X and Y are routing nodes, and nodes U and Z act as routing/proxy nodes.

Figure 2. First generation OR topology depicting a five hop OR circuit. Source: [15].

- Client and router were fully combined and all communication was done in a peer-to-peer (P2P) configuration. Syverson explains that this meant a client would have to use an application to proxy its traffic to an onion router if it did not actively participate in the Tor network as a node. If a client chose this option, its traffic would be sent to the first routing node before the actual circuit was built [5].
- Dynamic network updates and topology changes were not supported. It was thought that Tor communication had been between stable networks

that did not have inherent trust relationships and that all network configuration changes were handled offline [5].

- Because the Socket Secure (SOCKS) protocol was not widely adopted in the mid-1990s,[4] multiple application-specific protocol proxies were required to route the various types of information on the Tor network [5]. Further, each application-specific request required its own Transmission Control Protocol (TCP) stream which increased the volume of traffic and posed a possible security threat based on how many TCP streams were present [16].

(2) Second generation OR

- The client and router were separated, which allowed a client to learn about available nodes and build its own circuits without having to route traffic for others or blindly trust a remote node to build a P2P circuit as was required in first generation OR [5].

- Circuit lengths were no longer fixed; variable-length circuits up to 11 hops were supported within a single onion [5].[5]

- The SOCKS protocol was adopted for applications able to support it[5].

- A redirector feature was included. It forced all TCP traffic over the Tor network. By design, the redirector only worked on Windows NT and was not intended for public release [5].[6]

- OR entry and exit policies were enabled to allow organizations the ability to tailor the traffic for users who had access to their onion routers [5]. For example, an organization could set an exit policy to only allow forward web traffic on ports 80 or 443 or choose to whitelist trusted IPs and ports to minimize possible abuse. When implemented properly, entry and exit policies could prevent an onion router from appearing to be the source of malicious activity [16], [20].[7]

---

[4] The SOCKS protocol provides a standardized method for applications to communicate without the need for multiple application-specific proxies by uniformly encapsulating packets for both TCP and UDP (in version 5 only) applications. Conceptually, the SOCKS protocol operates between the application and transport layers [19].

[5] In the second generation of OR, a single onion could be constructed with encryption layers for up to 11 hops with an even greater length possible if tunneling techniques were used [5].

[6] According to Syverson, with the exception of the TCP redirector, "all other onion routing code of all generations" was intended for public release [5].

[7] Goldschlag et al. state, "Most onion routers in the current network function as restricted exits that permit connections to the world at large, but prevent access to certain abuse-prone addresses and services such as SMTP" [16].

- Two security features, cell mixing and traffic shaping, were implemented to lower the likelihood that someone would be able to associate user and activity through simple traffic analysis. Experimentation into cell mixing focused on blending TCP streams from separate onion routes to mitigate the threat of a man-in-the-middle (MITM) attack. Traffic shaping used an average of previous packet lengths to pad new traffic in an attempt to protect it from identification by simple traffic analysis. Neither cell mixing nor traffic shaping were included in follow-on generations [5].

3. Third Generation OR

The third generation of OR is commonly known as Tor. Tor provides the same basic protections as its predecessors; however, it affords much more protection. The increased protection can be attributed to how the virtual circuit was created and subsequently encrypted [5]. Tor's basic design uses three nodes to route obfuscated traffic between initiator and responder (see Figure 3).

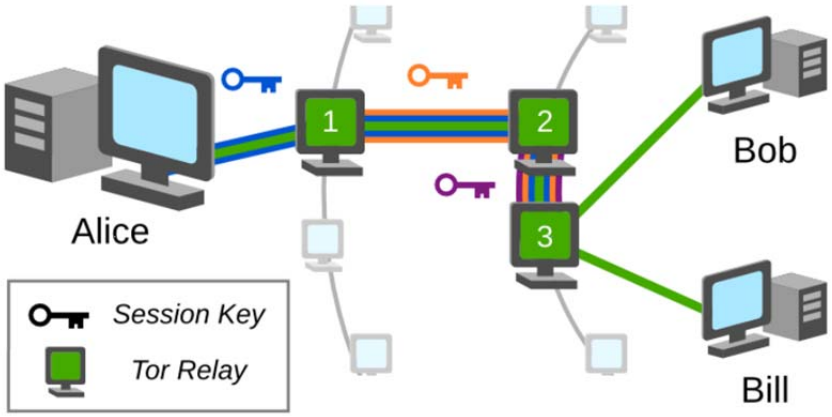
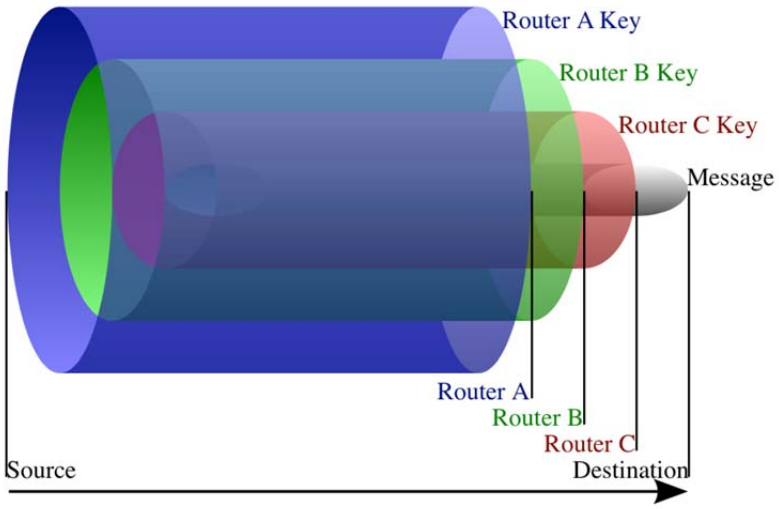


Figure 3. Basic Tor configuration. Source: [21].

Departing from the computationally expensive onion built using Public Key Infrastructure (PKI) encryption,[8] Tor uses DH key exchange-based circuit building and

[8] According to Syverson, PKI was used to encrypt the communication circuit in order to share symmetric keys from initiator to responder with an unpredictable route. The idea of using DH key exchange-based circuit building with RSA encryption was considered but abandoned during the first two generations of OR [5].

RSA encryption to construct the onion from hop to hop (see Figure 4) [5].[9] Further, since public ephemeral keys were used during the encryption process, the initiator could derive the session key by combining the ephemeral key with the private key of the responder (and vice-versa) to provide replay protection and forward secrecy [5], [18].



Routers A, B, and C represent the entry, middle, and exit nodes, respectively.

Figure 4. Diffie-Hellman based routing onion. Source: [23].

According to Dingledine et al., the SOCKS protocol is used in Tor to provide compatibility for most TCP-based applications with no additional modifications. Consequently, Tor relies on outside application-level proxies to provide application level services not compatible with SOCKS [16]. Additionally, Tor relies on a small set of trusted directory servers to distribute network status and information to routing nodes and clients.[10] Each server is owned and operated by independent parties and functions as a HTTP server. The HTTP server functionality allows available onion routers to upload their statuses and clients to download network state and listings of available routers. Each

[9] RSA is an asymmetric public key algorithm that facilitates encryption and digital signatures by using public and private keys, which are produced by generating large numbers from their prime factors. It is named after its creators Ron Rivest, Adi Shamir, and Leonard Adleman [22].

[10] In the current distribution of Tor, nine directory servers are hard coded into the browser package. The full listing is available from https://atlas.torproject.org/#search/flag:authority. Accessed 18 October 2016.

trusted directory server controls which nodes are allowed to join the routing network in an effort to reduce the risk of malicious routers entering the network [16]. Syverson acknowledges there are distinct gains and losses with the use of directory servers. Tor clients are allowed to obtain a listing of known onion routers that may be used to construct a circuit. However, Syverson contends that a limited number of trusted directory servers creates a network bottleneck [5].

B. TOR VULNERABILITIES

Due to Tor's widespread adoption and routing configuration, numerous vulnerabilities, or *attack vectors*, are inherently present in its design. Dingledine et al. gave significant consideration into how to mitigate the threat caused by existence of malicious actors who can monitor sizeable portions of the Internet. Though Tor was not specifically designed to withstand an Internet-scale attack, it was designed to prevent attacks using traditional traffic analysis [16]. Elahi et al. contend significant vulnerabilities exist because the network is comprised of volunteer nodes;[11] this is especially true if a malicious actor is able to control both the entry and exit node in a Tor circuit [14].

1. Malicious Guards

Elahi et al. revealed that each node can be bandwidth-weighted to increase the likelihood of selection. As a result, it is possible for a malicious actor to volunteer multiple, high bandwidth, entry and exit nodes in order to conduct end-to-end correlation of user activity. To break the malicious node "kill-chain," various fixes can be employed to lower the persistence of malicious entry nodes [14], [24]. The Tor Project (torproject.org) proposes multiple solutions including increasing the size of the Tor network and using fewer entry guards [24].

Although not fully resolved, the Tor Project admits there are flaws in the way the guards are selected [24]. It is difficult to establish a set of available guards without

[11]The Tor network is comprised volunteer routing nodes; each additional routing node increases the bandwidth and availability of the Tor network. Additional Tor FAQ are available at https://www.torproject.org/docs/faq.html.en, accessed 24 October 2016.

inclusion of malicious nodes. However, by scaling the network faster than the adversary can, the adversary's foothold will decrease, making attribution more difficult. Of note, the Tor Project contends that merely increasing the number of relays may not always decrease the risk of attribution; however, increasing the amount of entry nodes also increases the amount of nodes an adversary can monitor [24]. Further, torproject.org contends that the set of entry guards selected by a client can serve as a fingerprint, as another client is not likely to have the same set. This fingerprint could be used by an adversary to identify a specific client's obfuscated traffic. However, reducing the quantity of the guard set to two, increases the chance of collision between the two fingerprints, making attribution will be more challenging [24].

2. Directory Server Control

Tor uses a limited set of nine redundant trusted nodes that serve as directory servers.[12] Dingledine et al. emphasize that it is the responsibility of the directory servers to reach a network consensus and distribute it to all Tor clients.[13] Dingledine et al. define the four-step process used by the directory servers to reach a consensus. First, each directory server broadcasts its signed opinion of the network state to the pool of directory servers. Then, each directory server subsequently broadcasts all signed network states it has received. The second step ensures no directory server has signed more than one network state. In step three, each directory server combines the received network states with its own and broadcasts its signed version of the full network state. In step four, the directory servers rebroadcast the complete network consensus with signatures from all participating directory servers. To facilitate this level of information flow, each directory server also acts an HTTP server. This allows clients to request a listing of all available

[12] The full listing of directory servers is available from https://atlas.torproject.org/#search/flag:authority, accessed 24 October 2016.

[13] Biryokuv et al. define the network consensus as, "The list of all Tor relays is distributed by the Tor authorities in the [network] consensus document. The consensus is updated once an hour by the directory authorities and remains valid for three hours. Every consensus document has a 'valid-after' (VA) time, a 'fresh-until' (FU) time and a 'valid-until' (VU) time. The 'valid-after' timestamp denotes the time at which the Tor authorities published the consensus document. The consensus is considered fresh for one hour (until 'fresh-until' has passed) and valid for two hours more (until 'valid-until' has passed)."[10]

routers and their states, as well as other onion routers to upload their statuses. However, the limited number of servers also presents multiple vectors for attack [16].

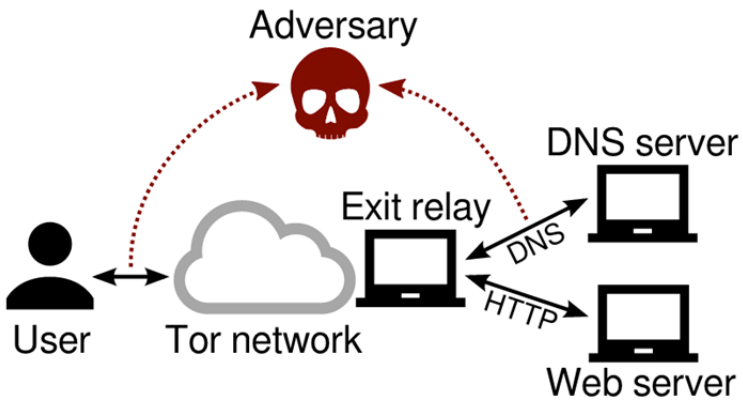
Dingledine et al. argue that if an adversary were to control a directory server, they could use the server to distribute a listing of malicious nodes under adversary control. The impact is much greater if an adversary can subvert control of a majority of directory servers because the adversary would have the ability to sign the network consensus with as many malicious nodes as it pleases. Additionally, if an adversary was able to either physically destroy a directory server or use other computer-based attacks they could render it useless, but simply taking down one directory server would not have a severe impact on routing [16]. However, if over half of the available directory servers can be taken offline, there will not be enough servers to sign the network consensus for distribution to clients. To mitigate these attacks, Dingledine et al. maintain it is necessary that each directory server is independently operated and hardened against computer-based attacks [16].

3. Leaking DNS Requests

According to research conducted by Greschbach et al., almost 40 percent of DNS requests by Tor users are observed by Google's DNS resolver; this poses a significant risk of attribution to Tor clients [20]. There are two ways DNS queries make Tor users vulnerable to attribution. First, as described by Dingledine et al., the SOCKS protocol, which is used heavily by Tor, can pose a risk to users. This is because some applications send the fully qualified domain name (FQDN) to the Tor client while others resolve the FQDN to an IP address before sending the request. The resulting threat is two-fold. First, an adversary monitoring DNS traffic from users could monitor what sites users are attempting to visit. Second, if an adversary has the ability to monitor or has control of a malicious exit node or group of nodes, the adversary may be able to correlate entire Tor sessions to a single user [16].

In a similar DNS-based attack, Greschbach et al. contend that if an adversary is able to monitor traffic before it enters the Tor network and DNS queries from the exit node, it may possible to correlate user and activity. This type of attack is called a "DNS-

enhanced fingerprinting and egress correlation on Tor," or a DefecTor attack (see Figure 5) [20].



In a DefecTor attack, the adversary must monitor DNS traffic entering and exiting the Tor network and is much more accurate if either or both of the entry guard and DNS Server are maliciously controlled.

Figure 5. Basic DefecTor DNS traffic attack. Source: [20].

As described by Greschbach et al., the DefecTor attack seeks to exploit the vulnerability posed by exit nodes resolving DNS queries outside the Tor network. The effectiveness of the DefecTor attack relies on two factors. First, on the adversary's ability to observe traffic entering the Tor network, and second, on how an exit node is configured to resolve and cache DNS queries.[14] Commonly, exit nodes are configured to resolve DNS queries by running either a local or third party (e.g., local ISP or Google) DNS resolver. Research by Greschbach et al. found that using a local resolver proved the most dangerous option leading to an increase in both the number of compromised Tor streams and days until first compromise. However, Greschbach et al. propose a different solution. In their study, Greschbach et al. observed that when only the ISP is used to resolve DNS queries, the probability of stream compromise is decreased while the

[14] In their research, Greschbach et al. discovered a bug in Tor's source code, referred to as TTL Clipping, which affects the way exit relays cache DNS lookups. By default, exit relays are capable of caching DNS queries between 60 seconds and 30 minutes, however, the bug discovered by Greschbach et al. resulted in all exit nodes caching DNS queries for 60 seconds. This increases the overhead on the network as well as opens a client up to an attack where the adversary needs to only wait 60 seconds between DNS queries [20].

number of days until first compromise is increased [20]. Other long-term solutions are configuring exit nodes to use Transport Layer Security (TLS) and TCP for DNS queries and,[15] for websites to offer Tor-based sites that do not require communication outside of the Tor network [20].[16]

C. IDENTIFICATION OF TOR TRAFFIC

Since 1996, researchers have focused on isolating identifiable characteristics of obfuscated low-latency network traffic such as Tor. Recent research has focused on identifying Tor traffic via traffic analysis [1], [2], [4], [6]–[10] HTTP flow analysis [4] entropy-based characteristics [1], [20] and semantics [1], [6], [11].

1. Traffic Analysis

Traffic analysis approaches have focused on examining the TCP/IP header information in each packet entering the network to infer specific information [25]. To do this, some method is needed to capture the packets as they enter the network; this can be done in multiple ways. First, by using a mirror port or switched port analyzer (SPAN), all data sent through a network switch can be captured. As described by Odom, while a SPAN or mirror port is traditionally used to monitor traffic within an internal network, it is possible to configure remote a SPAN (RSPAN) to monitor network traffic over a virtual LAN (VLAN) between layer-two switches and encapsulated RSPAN (ERSPAN) between switches operating at layer-three to capture network data between logically separated networks [26].[17] A second approach is to use network intrusion detection software (NIDS) such as Snort (www.snort.org), or Bro (www.bro.org). A NIDS is traditionally placed before network firewall and used to monitor all traffic entering a network [27]. To aid in traffic analysis, NIDS can be used notify system administrators of

[15] UDP stands for User Datagram Protocol. UDP is a connectionless protocol that is susceptible to higher loss than TCP. UDP is used heavily in applications able to tolerate packet loss (e.g., video streaming).

[16] "facebookcorewwi.onion" is an example of a website that does not require communication outside of the Tor network [20]. Accessed 18 October 2016 via TorBrowser.

[17] A layer 3 switch, also known as a multilayer switch, is a network device capable of operating at layer 2 for LAN switching and layer 3 for IP routing [25].

anomalous or suspicious activity and can be configured to produce packet captures (PCAP) containing a record of all data entering a network [26], [27].

Using a PCAP created by a NIDS, specific TCP/IP header data can be extracted and analyzed [25]. Based on research from [4], [6], [7], [9], [28], and [29], common obfuscated IP header fields of interest include time-to-live (TTL), total length, and source and destination addresses. Common TCP header fields of interest include sequence number, acknowledgement number, and the options block, which affect the overall size of the TCP header. The default IP and TCP header fields are depicted in Figure 6.

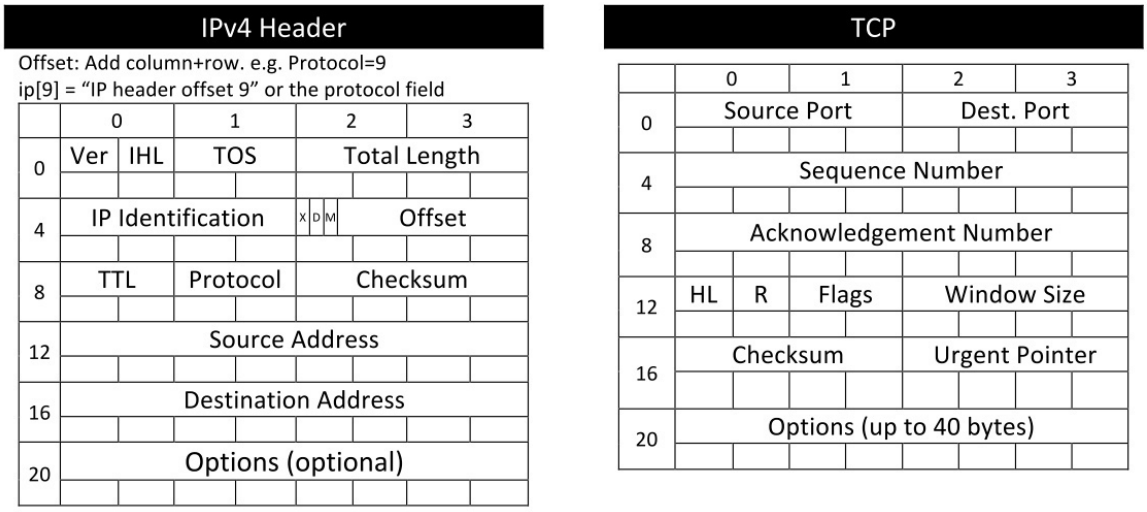


Figure 6. Default IP and TCP header fields. Source: [30].

a. Low TTL Count

Research by [13], [6], and [8] has observed a correlation between time-to-live (TTL), or hop-count, and the likelihood network traffic is spoofed or obfuscated. Further, as Kelly contends, by using the default operating system (OS) hop-counts and the average number of hops to traverse the Internet, it may be possible to determine probability of obfuscated traffic based on a lower TTL count [6].[18] Default TTLs are 128 for Windows,

[18] The TTL count of an IP packet is decremented by one by each routing device it passes through as it is routed to the destination. If the TTL count reaches zero, the packet will be dropped to prevent packets in an infinite loop.

64 for Linux, and 254 for Solaris-based operating systems.[19] Additionally, testing by Wang et al. showed the average hop-count on the Internet varies by the top level domain (TLD) visited [8]. Using *traceroute*, Wang et al. observed the following average hop-counts to TLDs:

- .com: 11 hops
- .edu: 4 hops
- .org: 2 hops
- .net: 12 hops

From the TLD down to actual websites, the average hop-count on the Internet was observed between 14 and 19 hops with 16 and 17 being the most commonly observed hop-counts [8].

b. High RTT Time

According to Syverson, users sacrifice some performance (increased overhead and bandwidth) for greater security due to Tor's architecture and routing schema [5]. This is somewhat offset by way of high-bandwidth nodes, but the multiple layers of encryption used by Tor inevitably results in higher latency and round trip times (RTT) [6], [14]. Kelly argues that based on the increased time it takes a packet to traverse the Tor network and the routing required outside of the network, higher RTTs may be indicative of obfuscated traffic [6].

Kelly contends the RTT difference can be measured by calculating the disparity between the timestamps of two packets. Related packets can be identified by their reversed socket-pair and TCP sequence and acknowledgement numbers [6].[20] This data can be extracted from a PCAP file produced by a NIDS. Shoukat et al. build on this idea observing a high correlation between TTL (hop-count) and RTT as shown in Figure 7

[19] Default TTL values for most modern operating systems are available from https://subinsb.com/default-device-ttl-values, accessed 13 October 2016.

[20] A TCP/IP socket is the combination of an IP address and TCP or UDP port. A socket pair is the unique combination of the sender and recipient sockets.

[28]. In a similar study, Claffy et al. observed a direct correlation between geographic separation and RTT (see Figure 8) [31].

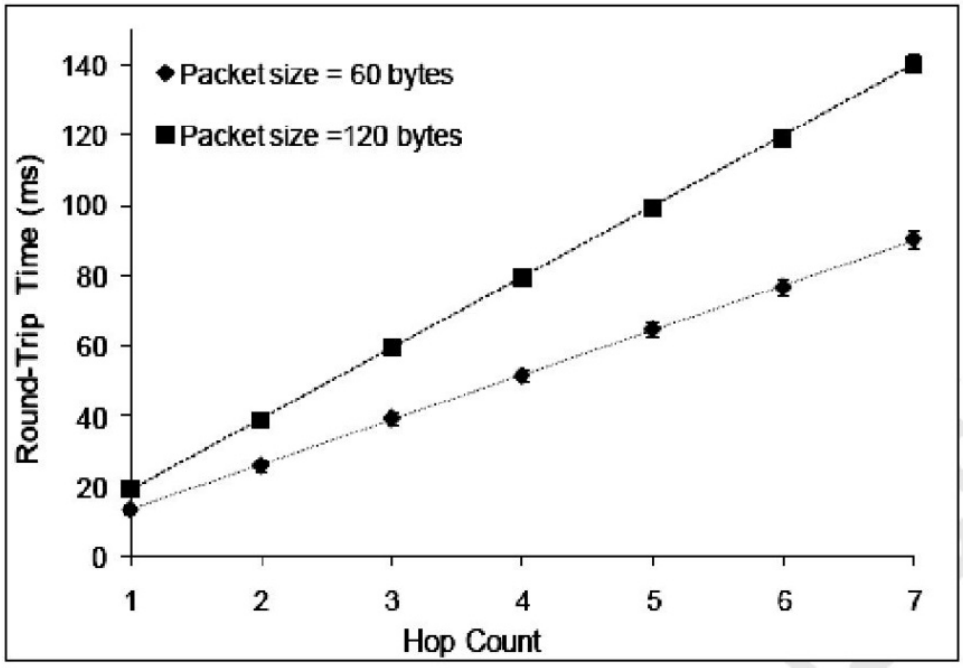
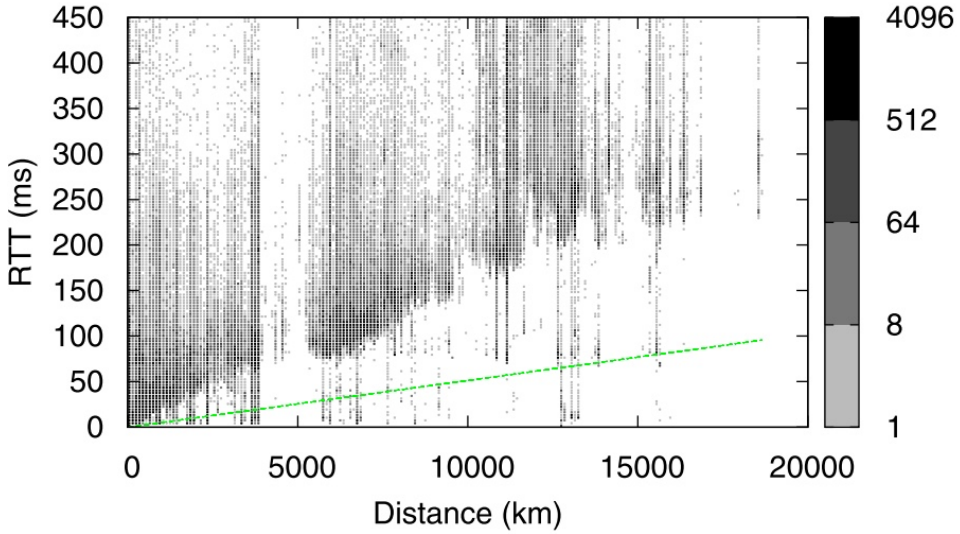


Figure 7. Linear relationship between hop-count (TTL) and RTT. Source: [28].

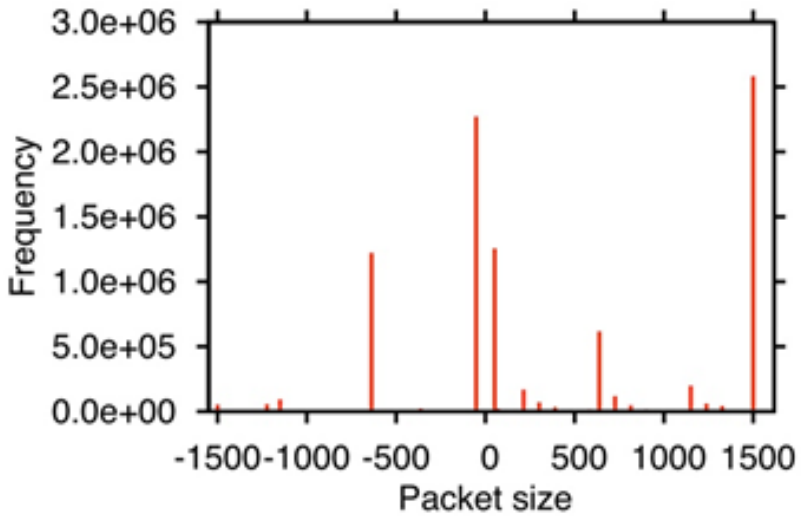


The scale on the left signifies the density of observed RTTs

Figure 8. Linear relationship between geographical distance and RTT. Source: [31].

c. Common Tor Packet Length

Using a Bayesian network fingerprinting technique, Herrmann et al. observed that 87.6 percent of Tor traffic exhibited a common packet size. In descending order of frequency, the most common Tor packet sizes are 1500, -52, -638, 52, 638, and 1150 bytes. Within the common Tor packet sizes, positive values indicate server to client communication and negative values indicate communication from client to server. Figure 9 shows the distribution of common Tor packet sizes. Herrmann et al. also contend that the remaining variations in observed packet sizes are caused by OS-specific fragmentation and that Tor's variation in packet size provides an additional level of protection as the false positive rate (FPR) using packet length alone for identification of Tor traffic would be prohibitively high [7].



Positive values indicate server to client communication and negative values indicate communication from client to server.

Figure 9. Common Tor packet size distribution. Source: [7].

d. High TCP Offset

Kelly contends that obfuscated traffic is suspected to have a higher TCP offset value due to Tor communication stream identification [9]. In addition to the required blocks in the TCP header, [16] emphasizes Tor uses the options field to include

additional routing information not usually observed in standard network traffic. As a result, the value in the offset field is higher than typical to account for the additional routing information [9].[21] To differentiate between obfuscated and normal network traffic, TCP payload offset values can be compared to determine offset deviation [9]. Figure 10 depicts a fully encapsulated packet portraying the locations of the TCP header and payload. If the additional TCP communication stream information is included in the TCP header, the payload will be physically shifted to the left to account for the extra header length.

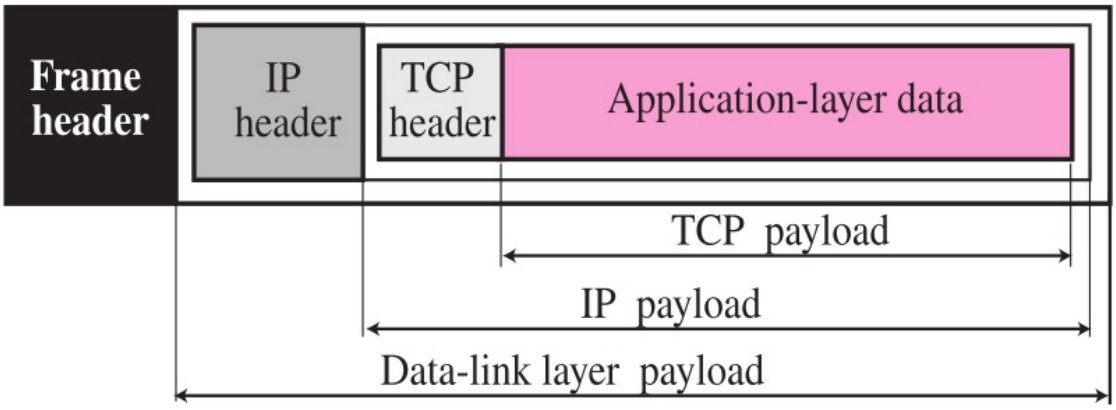


Figure 10. Fully encapsulated packet depicting TCP header and payload location. Source: [33].

2. HTTP Flow Analysis

Barker et al. observed a trend in HTTP and HTTPS flow sizes over the Tor network. By examining each TCP stream heuristically, packets three and five were shown to have a distinct size following the TCP three-way handshake.[22] Specifically, HTTP and HTTPS flow packets three and five fell between 131 and 152 bytes and 914 and 936

[21] The TCP offset field of the TCP header is called the data offset or header length (HL) as depicted in Figure 6. The TCP offset is the size of the TCP header in 32 bit words [32].

[22] Forouzan defines a three-way handshake as, "A sequence of events for connection establishment or termination consisting of the request, then the acknowledgment of the request, and then confirmation of the acknowledgment." The TCP three-way handshake follows the sequence defined by Forouzan using an exchange of packets between sender and destination with the following flags set: SYN, SYN-ACK, ACK [33].

bytes, respectively [4]. Table 1 specifies the packet sizes for the HTTP and HTTPS flow through the Tor network as observed by Barker et al.

Table 1. Observed HTTP, HTTP over Tor, and HTTPS over Tor flow packet size. Source: [4]

Index in Stream	Observed Packet Sizes
HTTPS	
0	0
1	0
2	0
3	100, 110
4	0
5	122, 516
HTTP Over Tor	
0	0
1	0
2	0
3	131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152
4	0
5	915, 916, 917, 918, 919, 920, 921, 922, 923, 924, 925, 926, 927, 928, 929, 930, 931, 932, 933, 934
HTTPS Over Tor	
0	0
1	0
2	0
3	131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 144, 145, 146, 147, 148, 149, 150, 151, 152
4	0
5	914, 915, 916, 918, 919, 920, 921, 922, 923, 925, 926, 927, 928, 929, 930, 932, 934, 935, 936

3. Entropy Based

Due to Tor's routing architecture, the receiving webserver sees the IP address of the exit node as the source IP for all packets [9], [16]. While research by Dingledine et al. focused on mitigating the third-party liability associated with running a Tor exit node by

way of exit policies [5],[23] Kelly argues exit node characteristics can be used to identify obfuscated traffic [9]. Specifically, the relationship between the source IP and associated session identifier may be indicative of an active Tor session.[24] Kelly's proposed correlation is dependent on a static session ID with a varying source IP address, both of which are affected by the maximum duration for a Tor circuit, which [5] defines as *maxcircuitdirtiness* time. According to Syverson, by default, each Tor circuit is used for a maximum of 10 minutes; however, users have the option of creating a new circuit when desired [5].[25] Another factor to consider is one from Greschbach et al. which shows that a Tor circuit remains open when at least one TCP stream is active, and the *maxcircuitdirtiness* timer resets each time a new stream is added [20].

4. Semantic Based

Syverson states the distributed exit node list provides Tor clients the necessary information to build OR circuits and can also serve as a blacklist if someone wants to block access from the Tor network [5]. While the list of active Tor exit nodes is updated every hour, Kelly's research found that of all detected Tor exit nodes (over 2,000), only half were listed on the distributed exit node list [6].[26] Kelly's finding affirms blacklisting Tor exit nodes is a necessary coarse filter but not sufficient for comprehensive detection of Tor traffic.

D. OTHER OBFUSCATION TECHNOLOGIES

Other obfuscation technologies similar to Tor make use of traditional Chaum mixes and the subsequent onion routing technologies based on its design. However, other technologies, such as protocol obfuscation, browser-based proxy websites, and decoy routing, use different methods to subvert censorship and gain access to the Tor network.

---

[23] OR Entry and exit policies are used to allow organizations the ability to tailor traffic for users who have access to their onion routers [5].

[24] The session identifier (sessionID) is used to identify and track all interactions between a browser and a webserver. Source: https://msdn.microsoft.com/en-us/library/system.web.sessionstate.httpsessionstate.sessionid(v=vs.110).aspx, accessed 14 October 2016.

[25] The *maxcircuitdirtiness* time can be modified in torrc, the configuration file [12].

[26] The hourly list of Tor exit nodes is available at https://collector.torproject.org/recent/exit-lists, accessed 13 Oct 2016.

1. Protocol Obfuscation

When combined, protocol obfuscation and traffic morphing allows a user to choose a source process by selecting a target process or set of processes. Researchers Wright et al. contend target processes can emulate a variety of common network traffic such as VOIP calls, HTTP traffic, and different types of applications [34]. In their study, Wright et al. showed traffic morphing could be used to change the packet sizes of an entire HTTP session to that of a target session from a different site. Specifically, they showed a download from *www.webmd.com* could be altered or morphed to appear as a download from *www.espn.com* by altering packet sizes in real time with minimal overhead [34].

SkypeMorph is an additional example of protocol obfuscation through traffic morphing. SkypeMorph obfuscates the communication between the client and the Tor network by using an established Skype call to conceal communication with an unlisted directory server or Tor bridge [35]. Tor bridges allow a client, who may be in an area that restricts access to the listing of known directory servers, to request the necessary information to create a Tor session without contacting a directory server [13]. Research by Moghaddam et al. has established that by using SkypeMorph, communication can be hidden in an established Skype video call with plausible deniability. This allows a client to subvert attempts to block access to the Tor network while appearing to communicate via Skype [35]. There are, however, some drawbacks to this method. Since high-speed communication is needed for video streaming, SkypeMorph uses UDP vice TCP to send information. As stated by Syverson et al., Tor traffic over UDP suffers from potential packet loss [5]. Further, [35] argues that since SkypeMorph uses only established video calls for communication, it is susceptible to constraints placed on bandwidth. The following obfuscation suites are offered, without detailed comparison, as other possible protocol obfuscation options: ScrambleSuite [36], Marionette [37], CensorSpoofer [38], and StegoTorus [39].

2. Browser-Based Proxy websites

Similar to the attempts made to block Tor directory servers and bridges, organizational or regional entities often block access to proxies that allow users to circumvent attempts at censorship.[27] Research by Fifield et al. offers a solution by way of browser-based proxy websites. To enable access, low bandwidth rendezvous protocols are used to provide a user with a list of available proxies. The proxies themselves are, ideally, innocuous volunteer websites outside the censored area. Each site agrees to place a small "Internet freedom" badge on its webpage (see Figure 11) [40].

Figure 11. Internet Freedom badge enabling access to browser-based proxy services. Source: [40].

The flash proxies created by [40] are fairly short lived, surviving only as long as the user is visiting the volunteer site. The ultimate goal is for all communications to appear as normal web traffic with authorized sites. Figure 12 illustrates the client-to-Tor relay process using a browser-based flash proxy [40].

[27] In 2009–2010 testing in Asia by the OpenNet Initiative found Burma, China and Vietnam had the most pervasive filtering of all tested Asian countries. The majority of the filtered content was independent media, politically sensitive or human rights information, and political reform sites [41].

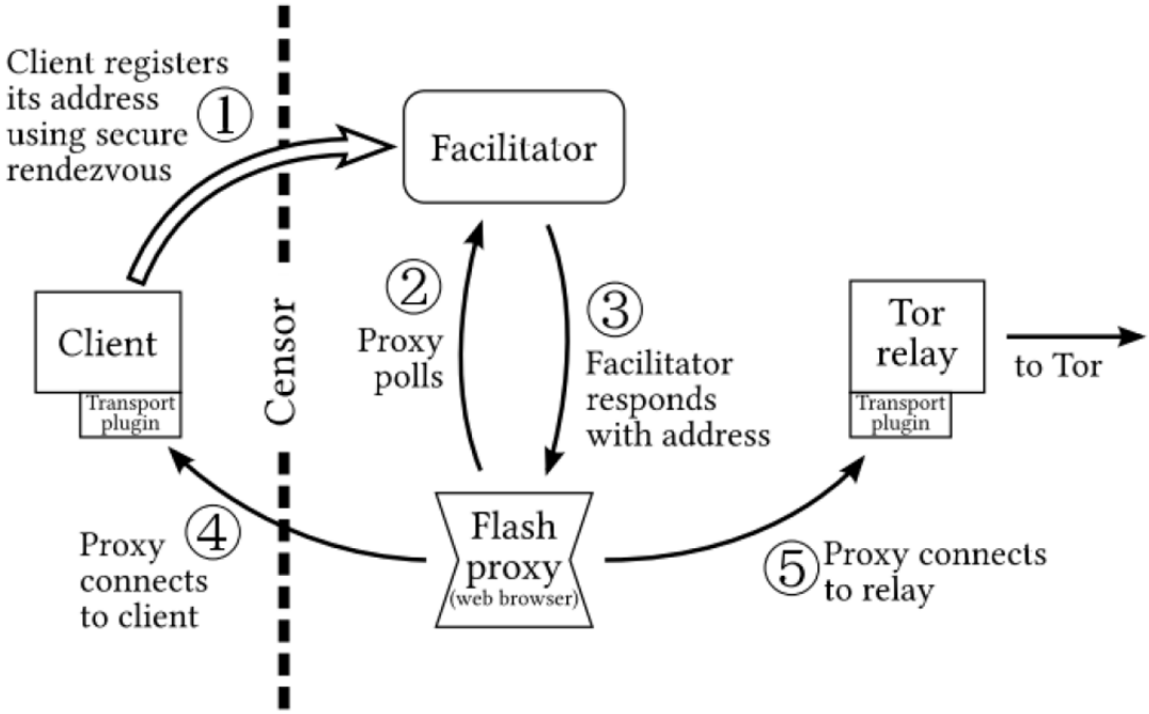


Figure 12. Simplified sequence of events from client to Tor relay using a browser-based flash proxy. Source: [40].

3. Decoy Routing

Decoy routing is another method that can be used to subvert attempts to block a client's access to the unauthorized content. Decoy routing differs from protocol obfuscation, traffic shaping, and browser-based flash proxies in that it is not designed to subvert attempts to block the Tor network. Instead, Karlin et al. contend that decoy routing is capable of avoiding traditional blacklisting techniques and can be used to allow client access to certain websites by way of decoy proxies [42]. To successfully circumvent traditional blacklisting, decoy routing leverages three foundational characteristics of the Internet. First, blacklisting routers is not feasible since their specific IP addresses are not included in the IP header information. Second, routing of packets on the Internet is federated, which leaves little routing control unless loose or strict source

routing is used.[28] Lastly, a decoy router can be placed in the virtual path between the client and numerous destinations, allowing it to serve a larger number of clients [42].

Karlin et al. describe the decoy routing process as follows. First, a client attempts to access an *allowed* website. Then, once the TCP handshake is complete, the client covertly signals to the decoy router to reroute the TCP flow to a decoy proxy. The decoy router communicates with the decoy proxy via the covert tunnel, which then accesses the blocked content (see Figure 13) [42].

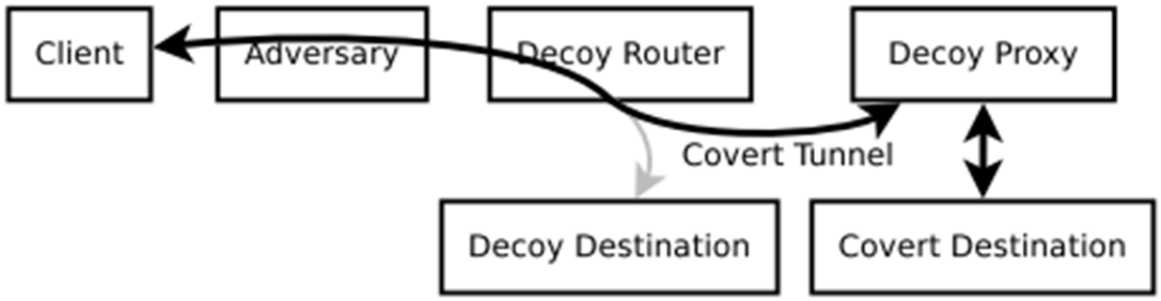


Figure 13. Decoy routing process from client to covert destination via a covert tunnel from the decoy router. Source: [42].

E. CONCLUSION

Obfuscation technologies offer users the unique ability to disassociate themselves from their online activity. While some may innocently use this technology, others could choose to use it in a malicious or unauthorized manner. To that end, multiple studies have investigated vulnerabilities and the identifiable characteristics associated with obfuscated traffic. The data processing requirements used to detect and exploit these characteristics has varied from study to study, but all required some level of advanced analysis and computation.

[28] When a circuit path is specified before any data is sent it is referred to as source routing. Source routing can either be strict or loose. In strict source routing, the path to the destination is explicitly defined and deviations are not allowed. Loose source routing is similar to strict source routing in that a path is specified, but differs because it allows deviations as long as every specified node is reached during transmission.

THIS PAGE INTENTIONALLY LEFT BLANK

III. METHODOLOGY AND TESTING

This research used a quantitative correlation approach to examine analysis techniques that could be used to identify low-latency obfuscated network traffic. To do this, real-world data was examined as it entered the network and analyzed using traditional network traffic analysis. This research approach and traffic analysis method systematically gathered data and allowed the examination of the relationships between characteristics associated with obfuscated traffic. Observing and defining each characteristic individually allowed for independent statistical analysis to determine its ability to function both as a single discriminator, and in a larger multi-attribute analysis model. During the multi-attribute testing, a Naive Bayes prediction model was used because of its ability to evaluate the data under the assumption that all variables are independent. As a result, the detection model tolerated individual characteristics with higher false-positive rates (FPRs) when attempting to detect obfuscated traffic.

A. DATA GENERATION

There are two ways to generate the necessary obfuscated traffic. The first is to use the Firefox-based Tor browser for all testing, and the second is to use a standard Firefox browser tunneled over an active Tor circuit.[29] The latter is possible by configuring Firefox to use the Tor circuit created by the Tor browser as a proxy; this configuration was verified but was not used during testing.[30] To generate non-obfuscated network traffic, a standard Firefox browser was used.[31] To standardize the obfuscated and non-obfuscated datasets, Selenium IDE, a Web browsing automation tool, was used to script browsing activity on the webserver.[32] Selenium was configured to execute a series of HTML commands to simulate normal browsing activity.

---

[29] http://www.wikihow.com/Use-Tor-With-Firefox provided step by step directions to manually configure Firefox to use an active Tor circuit. Accessed 15 November 2016.

[30] Manual proxy configuration is detailed in Appendix A.

[31] Baseline testing used Firefox browser version 50.0.1.

[32] Selenium IDE is a Firefox extension that is also compatible with the Tor Browser. A full description of Selenium IDE capabilities is available at http://www.seleniumhq.org/projects/ide/.

1. Baseline Data

Baseline data were required to evaluate and determine the performance of each indicator prior to constructing the multi-attribute detection model. Based on the assumed data requirements, only one iteration of baseline testing was required and conducted in two phases: obfuscated and non-obfuscated. The first phase generated and collected Tor-specific data, and phase two focused on non-Tor data. To collect the necessary amount of data, a combination of physical and virtual machines was used to gather 24 hours of Tor and non-Tor traffic, respectively. All data was written to Microsoft SQL database, and R-Studio was used to conduct statistical analysis after both phases were complete. Data analysis with R-Studio sought to determine each indicator's probability of detection.

2. Assumptions

To reduce the number of variables within the experiment, this research made the following assumptions.

1. Users will not adjust default Tor browser settings in the torrc.config configuration file, specifically the default 10 minute maxcircuitdirtiness timer setting.
2. Each hypothesized indicator will be observable in a dataset containing 24 hours of Tor and non-Tor network traffic.
3. Snort will log in real-time to Microsoft SQL database.

B. VIRTUAL LAB CONFIGURATION

The virtual lab environment was created on a Dell Power Edge R610 rack server running VMware ESXi 6.0.0. The server was configured with dual quad-core Xeon® E5620 2.4Ghz processors, 36GB of RAM, a RAID controller, and six 500GB hard drives, in a RAID 5 configuration. A Microsoft Windows 2012 R2 server was installed on one virtual machine and configured with the following software and services:

- SharePoint 2010
- Snort 2.9.2.2
- Wireshark 2.2.2
- Microsoft Internet Information Services (IIS) 8.5

- MSSQL 2014
- LogParser 2.2
- Active Directory

The virtual lab was configured with an Internet-facing SharePoint page, which enabled testing and evaluation of obfuscated and non-obfuscated network traffic for a variety of popular operating systems. Snort, Wireshark, and IIS logs were used to analyze the data generated by the SharePoint page. Specifically, Snort and Wireshark conducted network traffic analysis in real-time, and IIS logged server-dependent variables such as authenticated usernames. Snort was configured to log directly in MSSQL; however, Microsoft LogParser 2.2 was required to transfer the text-based IIS logs into MSSQL.[33] Wireshark was used to capture and save all testing data into packet capture–next generation (pcap-ng) files for further forensic analysis. Active Directory was used to create individual accounts for testing. Figure 14 provides an overview of the virtual lab.

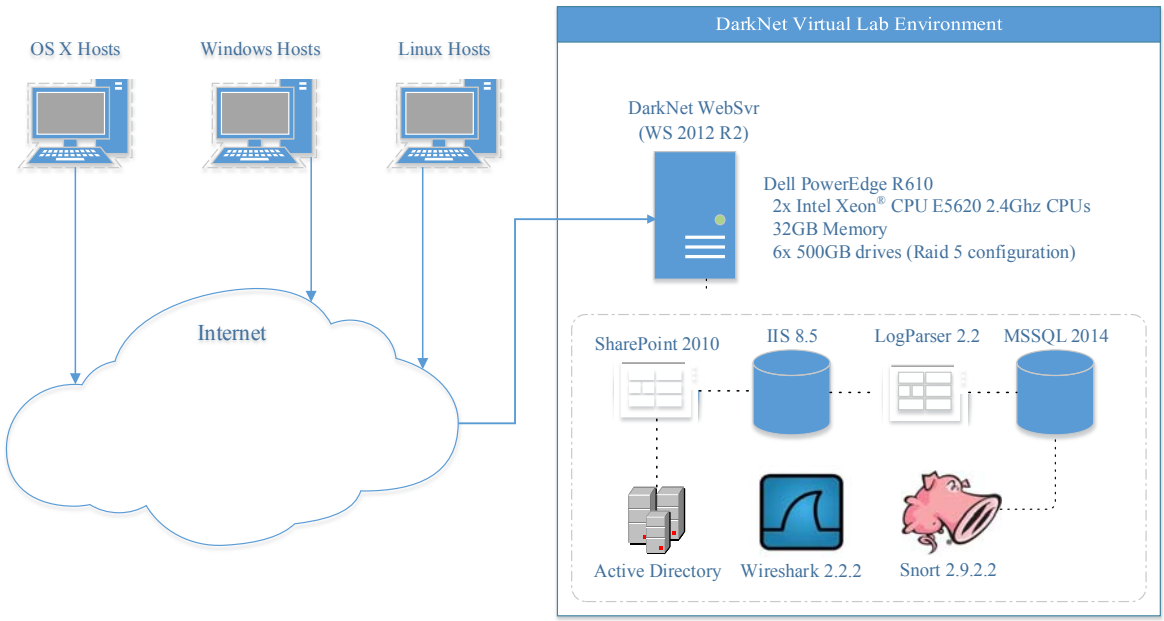


Figure 14. DarkNet virtual lab configuration.

[33] Snort and LogParser configuration settings are provided in Appendix B.

C. OBFUSCATION INDICATORS

This study used key attributes of four separate obfuscation indicators to determine whether network traffic was obfuscated. At an abstract level, the obfuscation indicators were referred to as a key cyber concepts (KCCs). Each KCC used either a pre-defined set of characteristics or an R-script to identify unique Tor and non-Tor attributes.[34] After analysis, unique attributes were labeled as obfuscated threshold values (OTVs) and used during testing to filter network traffic before processing to increase the probability of detection. Although each KCC is capable of independent analysis, KCCs One through Three were used concurrently to determine whether Tor traffic was discernable from non-Tor traffic within the multi-attribute prediction model.

1. KCC One: Low TTL Count

Using research by [6], [8], and [13], this research examined the time-to-live (TTL) field of the IP header to determine whether incoming traffic originated from Tor or routine network traffic. Further, since [43] observed that most Internet hosts are separated by no more than 30 hops, KCC One was tailored to examine the default TTL values for Linux (64) and Windows (128), which allowed sufficient separation between the initial hop counts during testing.

a. Required Data

All data required for KCC One was extracted from two separate Snort log tables located in MSSQL (see Table 2).[35]

[34] Each pre-defined set of characteristics derived from the current body of research described in Chapter 2.

[35] Required data descriptions for KCCs One through Three derived from [32].

Table 2. Required low TTL count data fields and source databases.

Data Field	Source Table	Description
timestamp	Event (event)	Specifies the time a packet was sent or received.
ip_src	IP Header (iphdr)	Specifies the IP address of the sending system.
ip_ttl	IP Header (iphdr)	Counter that is decrimented everytime the datagram passes through a network hop.

2. KCC Two: Common Tor Packet Sizes

This research attempted to detect common Tor packet sizes during testing to validate the observations by [7] that 87 percent of Tor traffic exhibited a common packet size of 1500, -52, -638, 52, 638, and 1150.[36] Additional Unique Tor and non-Tor packet sizes were also studied as both could be used as good discriminators in a multi-attribute decision model.

a. Required Data

Required data for KCC Two was extracted from two separate Snort log tables located in MSSQL (see Table 3).

Table 3. Required common Tor packet size data fields and source databases.

Data Field	Source Table	Description
timestamp	Event (event)	Specifies the time a packet was sent or received.
ip_src	IP Header (iphdr)	Specifies the IP address of the sending system.
ip_len	IP Header (iphdr)	Specifies the total length of the entire packet in 32-bit words.

[36] As described by [7], positive values indicate server to client communication and negative values indicate client to server.

3. KCC Three: High TCP Offset

Based on [9] and [16], this research assumed Tor traffic uses more TCP header options, which result in an observably higher TCP offset value than in non-Tor traffic. To verify the results of [9] and [16], this study analyzed the average TCP offset of Tor and non-Tor Web traffic to test whether it was possible to categorize each new instance of traffic into the webserver as either obfuscated or non-obfuscated.

a. Required Data

Data required for KCC Three was extracted from two separate Snort log tables located in MSSQL (see Table 4).

Table 4. Required TCP offset data fields and source databases.

Data Field	Source Table	Description
timestamp	Event (event)	Specifies the time a packet was sent or received.
ip_src	IP Header (iphdr)	Specifies the IP address of the sending system.
tcp_off	TCP Header (tcphdr)	The size of the TCP header in 32-bit words which indicates the starting point of the payload data.

4. KCC Four: Known Tor Exit Node

Torproject.org publishes its list of exit nodes hourly for node identification and blacklisting. Incorporating a coarse blacklist of Tor exit nodes can be a computationally efficient first step for identifying Tor traffic; this information is also useful for identifying exit nodes not on the published list. As a result, KCC Four was used as a blacklist to verify whether any network traffic originated from a known Tor exit node, which would indicate it was obfuscated.

a. Required Data

Data required for KCC Four was extracted from two separate Snort log tables located in MSSQL (see Table 5). Although not used, the data required was also logged and available in the Microsoft IIS logs.

Table 5. Required Tor blacklisting data fields and source databases.

Data Field	Source Table	Description
timestamp	Event (event)	Specifies the time a packet was sent or received.
ip_src	IP Header (iphdr)	Specifies the IP address of the sending system.

D. MULTI-ATTRIBUTE DECISION MODEL

Each KCC is capable of predicting the conditional probability that network traffic is obfuscated alone or in any combination to reduce both the FNR for Tor and FPR for non-Tor traffic. The multi-attribute model was used in an attempt to decrease the FPR by flagging packets which exhibit more than one of the possible attributes. To do this, the multi-attribute decision model uses a Naive Bayes classifier to accurately measure the cumulative conditional probability between each KCC. Figure 15 provides an overview of a simple Naive Bayes analysis two indicators and Figure 16 illustrates the multi-attribute decision model as a Venn diagram.

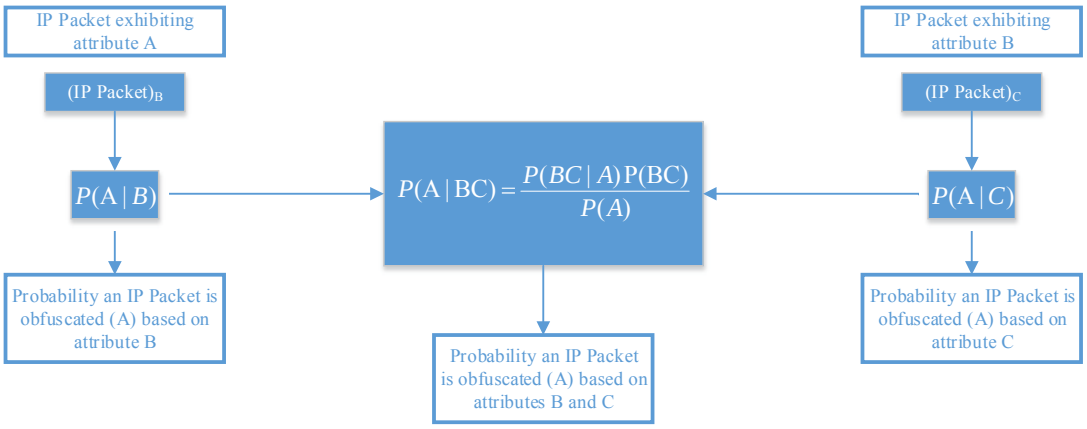
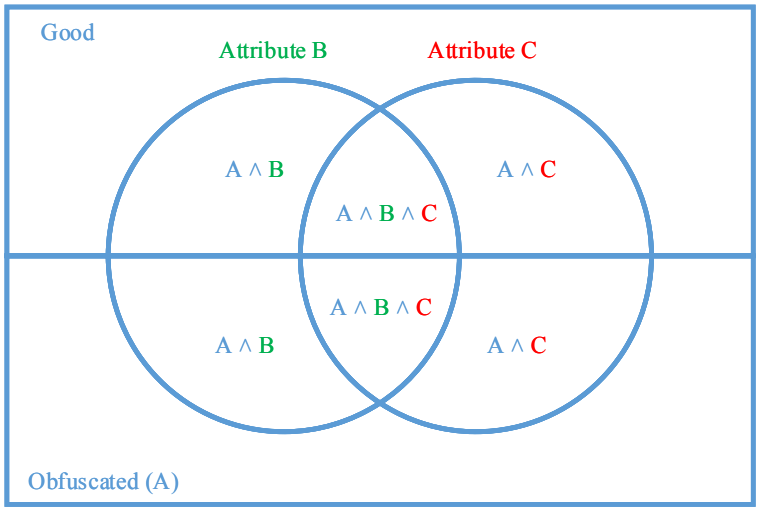


Figure 15. Notional multi-attribute decision model using Naive Bayes analysis.



Packets present in the lower hemisphere have a higher probability of obfuscation. The probability of obfuscation increases if packets exhibit more than one attribute.

Figure 16. Simplified Venn diagram illustrating the probability of obfuscation based on two attributes. Source: [44].

E. CONCLUSION

Using traffic analysis is a simple approach capable of identifying unique characteristics present in a TCP/IP network traffic dataset. Those characteristics can then be used as a queue for advanced analysis or any other administrator-defined process. This

study also sought to increase compatibility across different networks by using commercial off-the-shelf (COTS) hardware and software suites throughout the testing and analysis. Further, examining and testing each indicator with R-script and Naive Bayes prediction models allows maximum flexibility when replicating this research or applying it to real-world networks and applications.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. DATA ANALYSIS AND RESULTS

A. DATA ANALYSIS

Analysis of 702,376 unique packets of Tor and non-Tor data showed distinct attributes for all indicators, or key cyber concepts (KCCs), which were viable discriminators in a multi-attribute analysis model.[37] Before using the observations in a multi-attribute analysis model, each KCC was individually tested both with and without filtering for packets exhibiting the observed attributes to determine its ability to discern Tor traffic. The testing was conducted in two ways. First, using the list of attributes as a pre-filter, the datasets were filtered to include only those rows that exhibited the observed attribute, and second, the datasets were evaluated in their unfiltered states. After individual testing, each KCC's attributes were applied in a multi-attribute analysis model in an effort to increase the probability of detecting Tor traffic. Similar to the individual testing, the model evaluated filtered and unfiltered datasets. The R-script used for evaluation and testing appears in Appendix D.

1. KCC One: Low TTL Count

Analysis showed 56 unique IP time-to-live (TTL) values present in Tor traffic and 77 unique TTL values present in non-Tor traffic. Interestingly, most Tor TTL values were concentrated between 30 and 60 with very few observations above 100. In contrast, non-Tor TTL values were also observed between 30 and 60, but many exhibited a value above 60, albeit in lower quantities (see Figure 17).

[37] Both the Tor and non-Tor dataset were comprised of 351,188 unique rows of data.

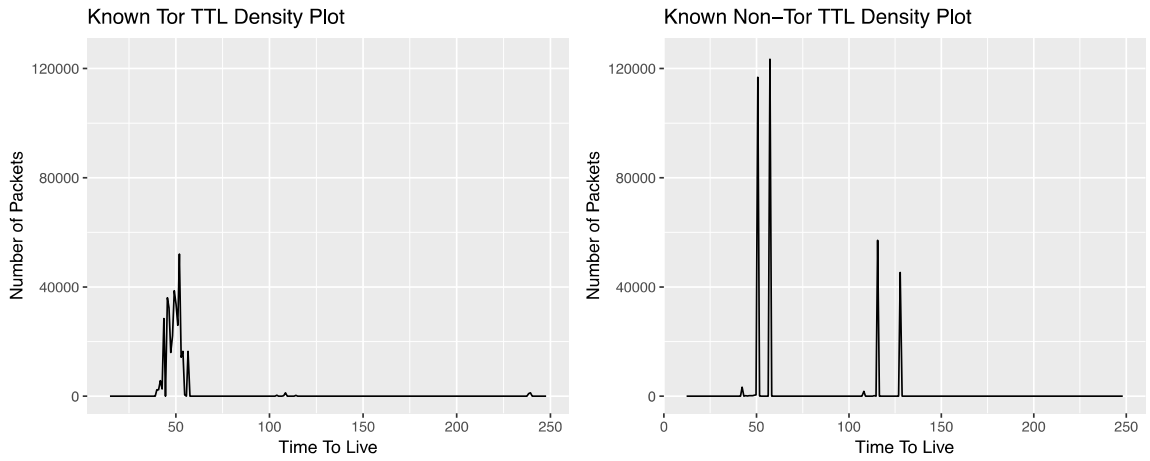


Figure 17. Density of Tor and non-Tor TTL values 0–250.

The observed Tor TTL characteristic is consistent with the default operating system TTL count of 64, which is normally attributed to Linux-based systems. According to Tor exit node data from January 2017, Linux-based operating systems accounted for 91.3 percent of all Tor exit nodes.[38] Figure 18 illustrates the observed concentration of Tor TTL values under 60 compared to non-Tor traffic. Notable concentrations were observed in Tor traffic at 44, 45, 46, 47, 48, 49, 50, 52, 53, and 54 while non-Tor IP TTL values were concentrated at 51, 57, 116 and 128. Confirming these unique TTL values enables their use as discriminators in both single and multi-attribute analyses.

[38] Tor exit node OS data is available from http://torstatus.blutmagie.de/index.php, accessed 18 January 2017.

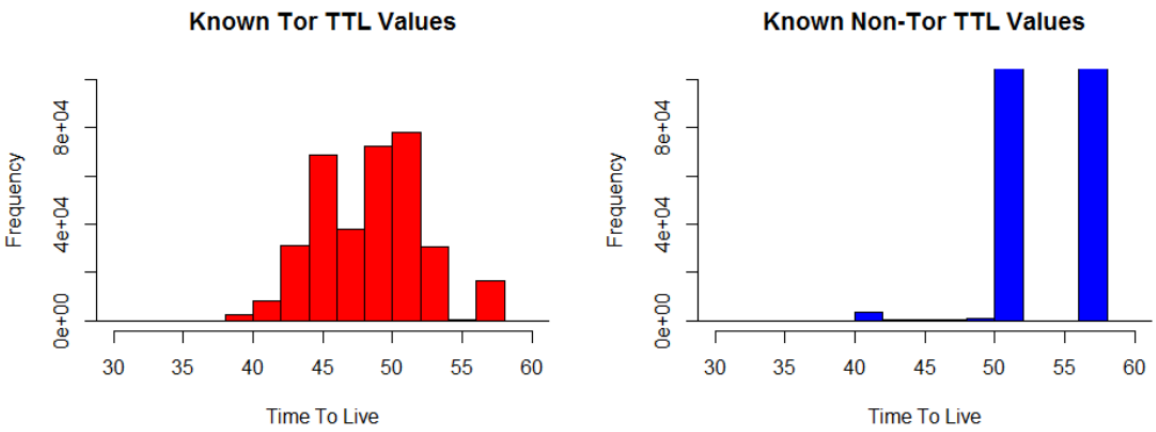


Figure 18. Concentration of Tor and non-Tor TTL values 30–60.

Further analysis examined the mean and standard deviation of both datasets. While the mean and standard deviation may not solely detect obfuscated activity, they can be used to quantify the overall heuristics of Tor and non-Tor traffic. The calculated mean and standard deviation of Tor traffic and non-Tor support are provided in Table 6. Of note, Tor's calculated mean and standard deviation support the assertion that most exit nodes are using Linux-based operating systems with a default initial TTL value of 64.

Table 6. Observed IP TTL mean and standard deviation.

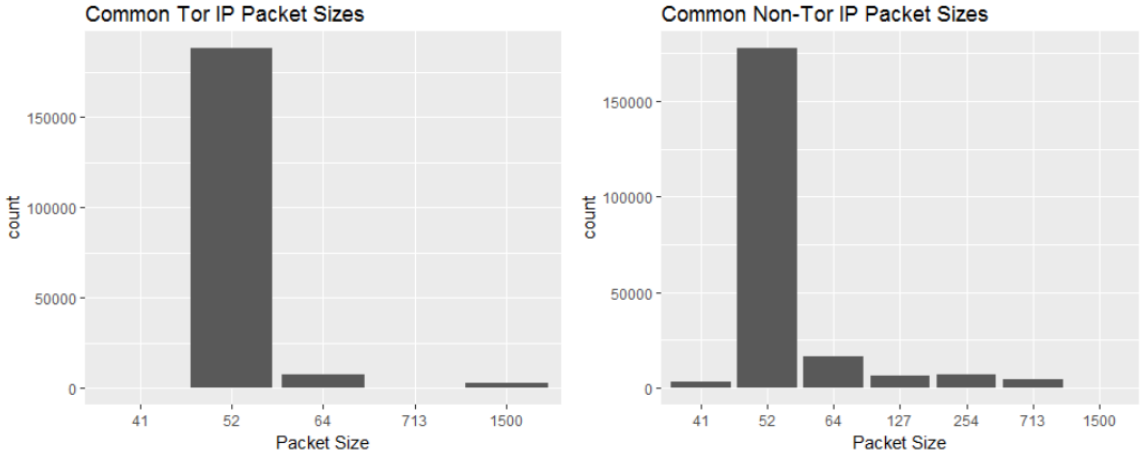
IP Time-To-Live	Tor	Non-Tor
Mean	50.68	74.34
Standard Deviation	16.95	32.09

2. KCC Two: Common Tor Packet Size

Analysis of Tor and non-Tor packet sizes began by examining the data to test the results of [7], who observed common Tor traffic IP packet sizes of 52, 638, 1150, and 1500. However, after analysis, Tor packets were observed only at sizes 52 and 1500 and accounted for 54.5 percent of all Tor packet sizes. However, additional analysis observed a low number of Tor and non-Tor packets with a size of 1500 and a very high number of

both exhibiting a packet size of 52. Specifically, packet size 52 accounted for 52.1 percent of all combined Tor and non-Tor traffic, and packet size 1500 accounted for only 0.4 percent of the same dataset. Thus, solely using packet sizes of 52 and 1500 will not serve as a good discriminator in either single attribute or multi-attribute analysis.

Additional examination of the entire range of packet sizes 40–1500 showed approximately 77.3 percent of all Tor and non-Tor traffic exhibited packet sizes of 40 and 52. Specifically, 81.4 percent of Tor traffic and 73.2 percent of non-Tor packets exhibited either a packet size of 40 or 52. As a result, identifying Tor and non-Tor traffic using packet sizes 42 and 50 was a statistical coin flip. In the remaining traffic, additional packet sizes were manually reviewed to discern which ones could be used as effective discriminators. Unique Tor packet sizes were observed at 638, 1150, and 1500, and unique non-Tor packet sizes were observed at 41, 64, 127, 254, and 713 (see Figure 19). Since the unique TCP offset values were only observed in a small number of packets, multi-attribute analysis would likely be more effective than single attribute analysis at detecting Tor traffic.



Packets with a size of 52 were included to illustrate the small number of unique packet sizes observed. Of note, packet lengths 127 and 254 were not observed in the set of known Tor IP packets.

Figure 19. Unique Tor and non-Tor IP packet sizes.

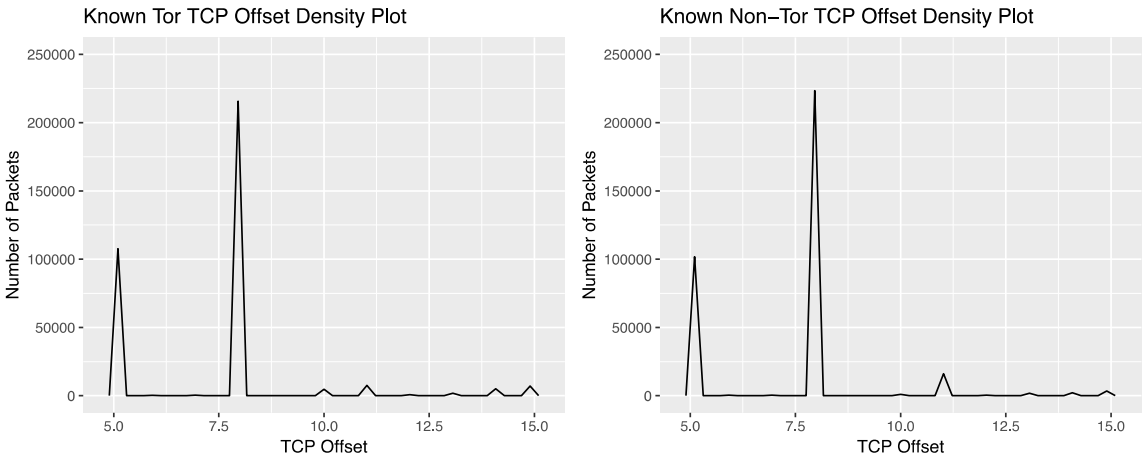
Similar to the observations for IP TTL, Tor IP packet sizes exhibited a lower mean and less deviation than non-Tor packets. This may suggest the Tor network fragments the packets into a lower number of small packets sizes; however, based on the size of the dataset and the small disparity between the observed mean and standard deviations, additional analysis is recommended (see Table 7).

Table 7. Observed IP packet size mean and standard deviation.

IP Packet Length	Tor	Non-Tor
Mean	103.5	122.7
Standard Deviation	202.26	215.79

3. KCC Three: High TCP Offset

Analysis showed that 92 percent of Tor packets and 92.6 percent of non-Tor packets exhibited a TCP offset value of either 5 or 8 (see Figure 20). Based on this observation, only TCP offset values greater than 9 could be viable discriminators between Tor and non-Tor traffic.



In both Tor and non-Tor traffic, an offset value of 8 was the most common value between 5 and 15.

Figure 20. Tor and non-Tor TCP offset density values 5–15.

In examining only the data with TCP offset values greater than 9, offset values of 10, 12, 13, 14, and 15 were observed in approximately 5.6 percent of Tor packets while the same values were observed in only 2.7 percent of non-Tor packets. Further analysis showed that a TCP offset value of 11 was present in 4.6 percent of non-Tor packets as compared to only 2.2 percent of Tor packets. Combined, TCP offset values greater than 9 represented 7.4 percent of all Tor and non-Tor packets (see Figure 21). Similar to the assumptions made after examining IP packet length, using unique TCP offset values represents such a small amount of the overall dataset that they may be more effective when used in multi-attribute analysis.

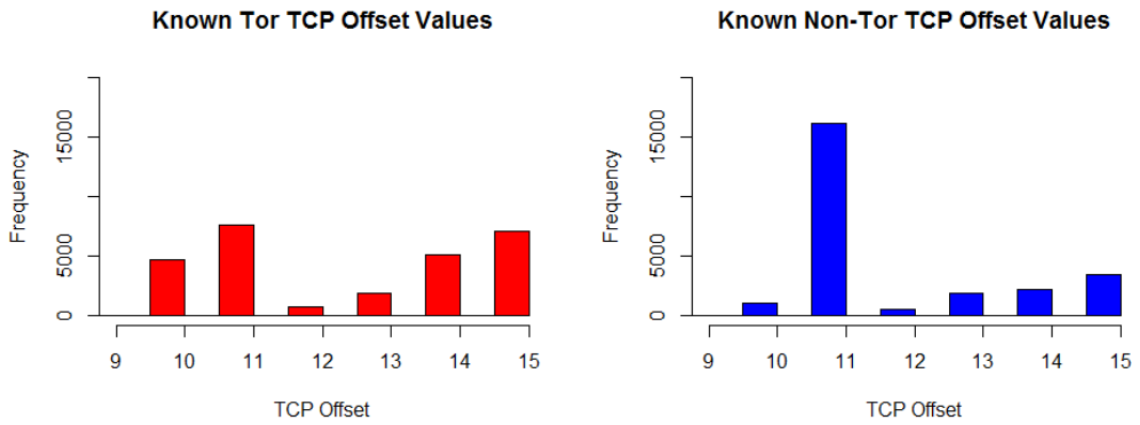


Figure 21. Tor and non-Tor TCP offset values 9–15.

Unlike the characteristics observed with the IP TTL and packet length, there was very little difference between the mean and standard deviation of Tor and non-Tor packets. Both exhibited a nearly identical mean of 7.4 and only showed minimal separation between their standard deviations (see Table 8). This suggests the additional TCP header data required by Tor has a negligible effect on the overall TCP offset.

Table 8. Observed TCP offset mean and standard deviation.

TCP Offset	Tor	Non-Tor
Mean	7.43	7.41
Standard Deviation	2.09	1.89

4. KCC Four: Known Tor Exit Node

To verify the accurate coverage of the distributed Tor exit node list, this research used R-script to identify unique IP addresses in both Tor and non-Tor datasets separately. The unique rows were then compared to a listing of known Tor exit nodes active during the testing period. Results showed 1,218 unique Tor IP addresses, 2,091 unique non-Tor IP addresses, and 1,106 unique Tor exit node IP addresses. Each set of unique addresses was compared to identify intersections, or collisions, with the other datasets. Interestingly, there were only 178 intersections between the known Tor IP address dataset and the known Tor exit node dataset. This number accounted for only 14.6 percent of the source IP addresses in the known Tor dataset. Based on these results, this study has replicated the observations by [9] that a majority of known Tor traffic did not originate from a published exit node during the testing period. Further comparisons showed zero intersections between the known Tor exit node dataset and known non-Tor dataset. Based on these two data points, when packets originate from a known Tor exit node, they are obfuscated.[39]

5. Inter-Attribute Correlation

Analysis of KCCs One through Three resulted in the identification of common characteristics shared by Tor traffic. However, this study also observed that each packet is not likely to exhibit all traits at the same time. To measure the inter-attribute correlation, three filters were used.

[39] Of note, there were 96 intersections between the Tor and non-Tor IP address datasets and accounted for 2.9 percent of the total unique IP addresses; this was likely due to non-Tor addresses accessing the Internet-facing website during Tor testing.

1. Filter one selected only packets that exhibited a TTL between 43 and 60, with the exception of packet sizes 52, 56, or 57.

2. Filter two selected only packets that exhibited a TCP offset value of 10, 11, 12, 14, or 15.

3. Filter three selected only packet sizes 41, 52, 64, 127, 254, 713, or 1500.

Figure 22 illustrates the negligible inter-attribute correlation among the first three KCCs after the filters were applied. The correlation among attributes is indicated using decimal values in the upper right and the red lines contained in the boxes on the lower left corner of the plot. The extremely low decimal values and horizontal red lines indicate very little to no correlation among the attributes examined. However, based on the limited size of the dataset, correlation among the studied variables cannot be entirely discounted.

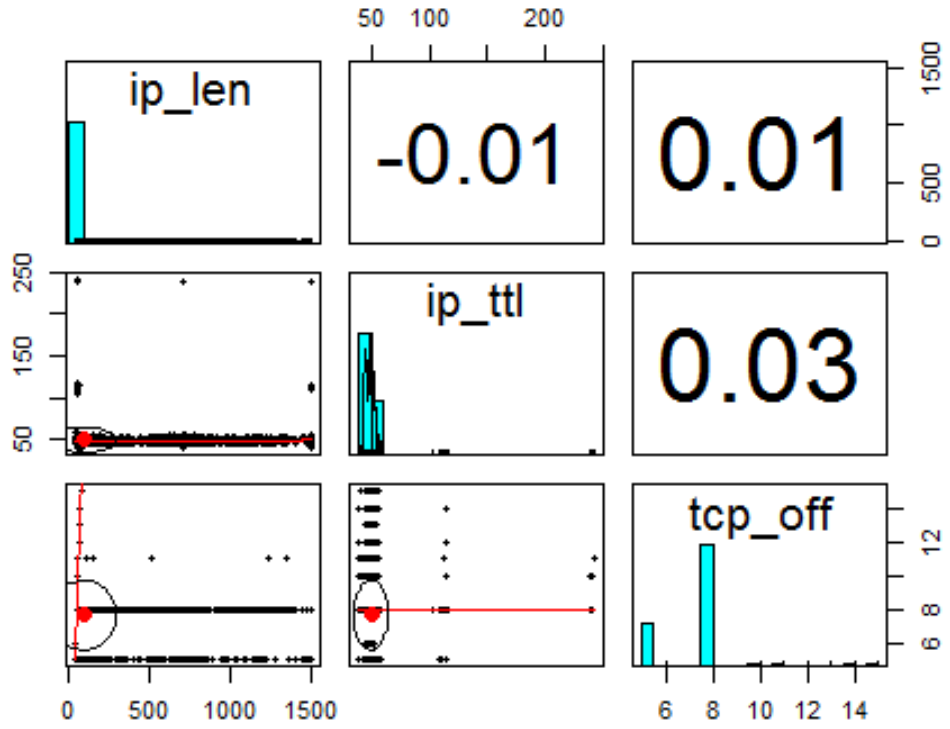


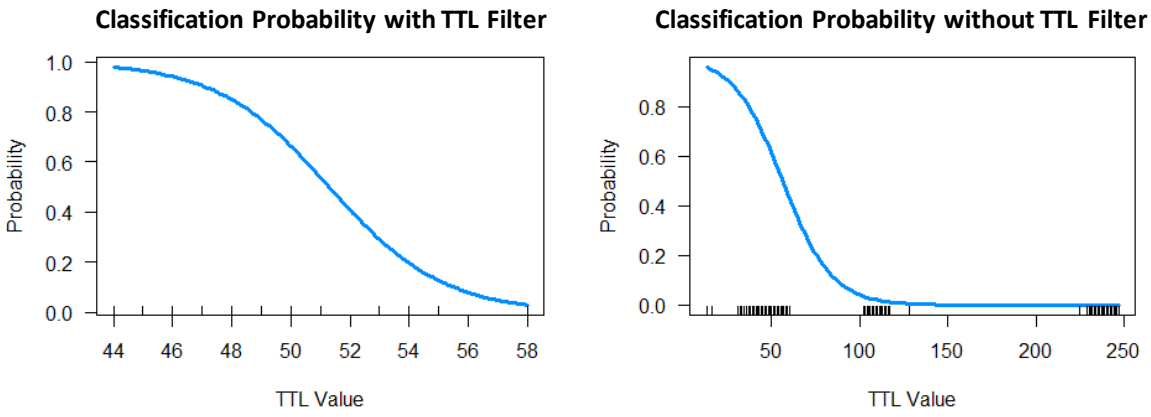
Figure 22. Filtered inter-attribute correlation of KCCs One through Three.

B. SINGLE ATTRIBUTE ANALYSIS

Each KCC was assessed to determine its ability to discern Tor from non-Tor traffic. For single attribute analysis, Tor and non-Tor datasets were differentiated with a 1 or 0, respectively, in an additional column. After the datasets were marked, they were combined and analyzed using R-script to determine the probability of correctly identifying Tor traffic.

1. KCC One

The known Tor dataset was evaluated with and without filtering for common Tor IP TTL characteristics. After filtering and analysis, an empirical misclassification rate of 39.1 percent was observed when filtering for TTL values between 43 and 60, with the exception of packet sizes of 52, 56, or 57; the unfiltered dataset returned a misclassification rate of 20.4. The unfiltered misclassification rate was lower likely due to the inclusion of TTL values above 64, which are not likely to originate from a known Tor exit node. Based on the results observed, KCC One could be an effective coarse indicator of Tor traffic. Figure 23 illustrates the probability of evaluated packets being Tor based on IP TTL.

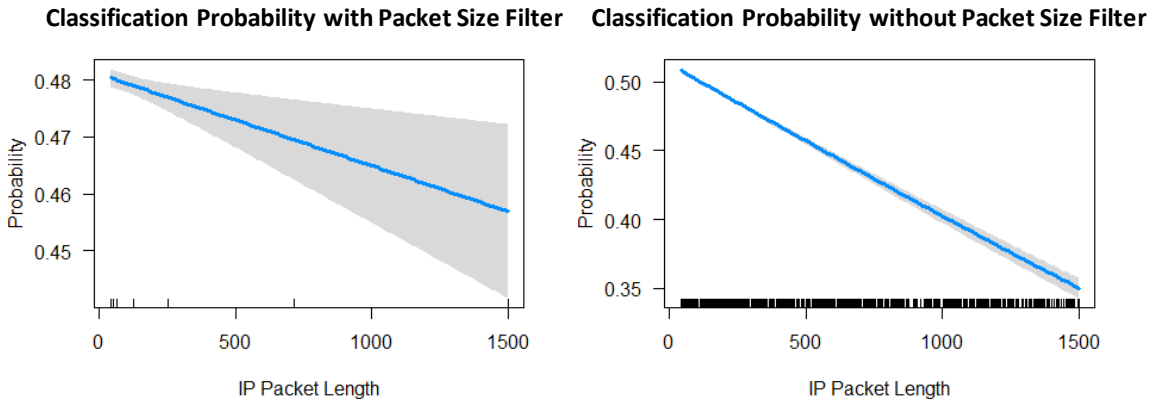


The left plot is filtered to examine only packets with a TTL value between 43 and 60, with the exception of packets sizes 52, 56, or 57; the right plot is unfiltered.

Figure 23. Classification probability of Tor traffic based on IP TTL.

2. KCC Two

The known Tor dataset was evaluated with and without filtering for common Tor IP packet size characteristics. Both the unfiltered dataset and the filtered dataset containing only packet sizes of 41, 52, 64, 127, 254, 713, or 1500 both resulted in an empirical misclassification rate of 47 percent. Based on the observed results, KCC Two would likely be an ineffective detector of Tor traffic when used alone. The results suggest that discerning Tor from non-Tor traffic would be a statistical coin flip. While Tor traffic did exhibit some unique packet sizes, the number of common Tor and non-Tor packet sizes far exceeds the number of unique sizes. Figure 24 illustrates the probability of evaluated packets being Tor based on IP packet size.

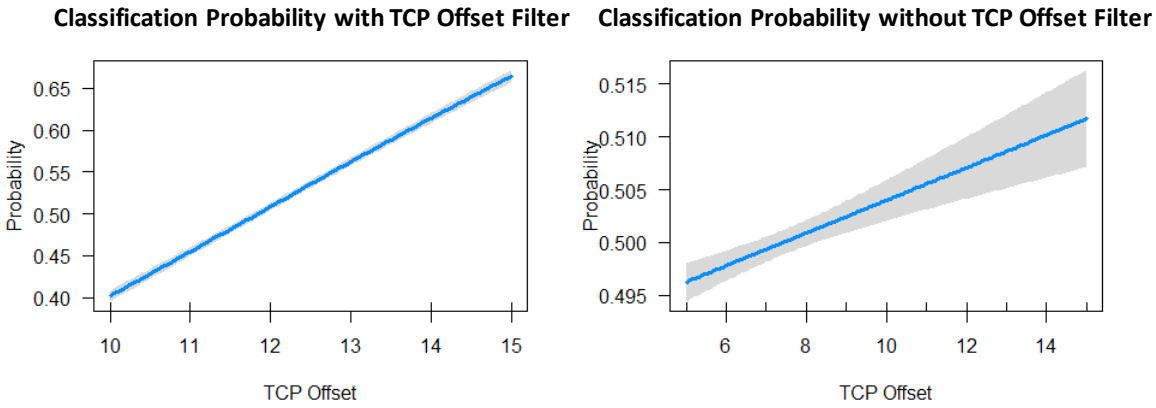


The left plot is filtered to examine only packet sizes size of 41, 52, 64, 127, 254, 713, or 1500; the right plot is unfiltered. The shaded area in the plot indicates the 95 percent confidence range of probability for any given value on the x-axis.

Figure 24. Classification probability of Tor Traffic based on IP packet size.

3. KCC Three

The known Tor dataset was evaluated with and without filtering for common Tor TCP offset values. An empirical misclassification rate of 37.9 percent was observed when filtering for TCP offset values between 10 and 15, with the exception of 13; the unfiltered dataset returned a misclassification rate of 50.8 percent. Based on the observed results, KCC Three could be used a coarse indicator of Tor traffic. Figure 25 illustrates the probability of evaluated packets being Tor based on TCP offset.



The left plot is filtered to examine only packets exhibiting a TCP offset greater than 9; the right plot is unfiltered. The shaded area in the plot indicates the 95 percent confidence range of probability for any given value on the x-axis.

Figure 25. Classification probability of Tor traffic based on TCP offset characteristics.

C. MULTI-ATTRIBUTE ANALYSIS

Ten tests were conducted to test the effectives of the multi-attribute analysis using the observed characters of KCC One through Three.[40] First, a Naive Bayes prediction model was trained with a combined dataset in two different ways: filtered by the observed characteristics and entirely unfiltered. The filtering was conducted in three configurations: first, by IP TTL, IP Packet Size, and TCP offset; second, by IP TTL and IP packet size; and third, by IP TTL only. Each filter used the observed threshold values (OTVs) detailed in the previous section of this chapter. After filtering, each Naive Bayes model used all three variables to analyze filtered and unfiltered test datasets.[41]

1. Baseline Testing

To establish a performance baseline, Naive Bayes analysis was conducted using an entirely unfiltered training and test set. The test resulted in a false-negative rate (FNR) of 6.2 percent for Tor traffic and non-Tor false-positive rate (FPR) of 60.4 percent (see Table 9). While the observed FPR was prohibitively high, the data were not filtered

[40] KCC Four was excluded from the multi-attribute analysis because of the observations during analysis that when traffic originates from a Tor exit node it is most likely Tor.

[41] The test dataset was filtered with the same filter used to train the Naive Bayes classifier.

before analysis. As a result, Tor and non-Tor packets not exhibiting the observed common Tor characteristics were included in the analysis.

Table 9. Naive Bayes unfiltered training and unfiltered test results.

Unfiltered Training and Unfiltered Test	Tor	Non-Tor
Packets Classified Correctly	82376	33162
Packets Misclassified	5420	50634
False Positive Rate		60.4%
False Negative Rate	6.2%	

2. Filtered Training

With baseline testing complete, each follow-on test used identical datasets to standardize the results. Using three pre-filter variations, the observed FPR for non-Tor ranged from 94.4 percent to 7.2 percent, and the observed FNR for Tor ranged from 61.3 percent to 1.6 percent. Interestingly, the tests that used more filters performed poorly.[42]

The best performing test used only the IP TTL to filter the datasets and reduced their overall size by approximately 55.6 percent. Specifically, 36 percent of Tor packets and 53.3 percent of non-Tor packets were removed by the filters.[43] The training set's initial a priori probability was 72.4 percent for Tor and 27.6 percent for non-Tor traffic. After analysis, the IP TTL–filtered test resulted in an FPR of 7.2 percent for non-Tor and an FNR of 8.7 percent for Tor (see Table 10). However, by manually accounting for the 31,558 known Tor packets excluded by the filter the overall FNR for Tor increased to roughly 41.5 percent. The FNR could be reduced by examining the data by source IP rather than by packet-by-packet analysis.

[42] Results of all Naive Bayes multi-attribute tests is provided in Appendix C.

[43] Before filtering, there were 87,796 Tor packets and 83,795 non-Tor Packets present in the test dataset. The filter removed 31,558 known Tor and 44,646 non-Tor packets from the original test dataset.

Table 10. Naive Bayes IP TTL filtered training and test results.

Filtered Training and Filtered Test	Tor	Non-Tor
Packets Classified Correctly	51353	36343
Packets Misclassified	4885	2807
False Positive Rate		7.2%
False Negative Rate	8.7%	

The next best overall results were observed using the same IP TTL filter on the training set but evaluating an entirely unfiltered test set. This test observed an FPR of 21 percent for non-Tor and an FNR of 20 percent for Tor traffic (see Table 11). While these results were significantly higher than in the preceding test, no Tor packets were filtered before analysis resulting in an 80 percent probability of detection.

Table 11. Naive Bayes IP TTL filtered training and unfiltered test results.

Filtered Training and Unfiltered Test	Tor	Non-Tor
Packets Classified Correctly	70218	66211
Packets Misclassified	17578	17585
False Positive Rate		21.0%
False Negative Rate	20.0%	

D. CONCLUSION

The study of each KCC did result in either the confirmation of previously observed indicators or in new identifiable characteristics. Furthermore, because this study focused on identifying Tor traffic, defined filters were capable of removing packets that

did not exhibit the observed common characteristics of Tor traffic. While this did filter out over half of the known non-Tor packets, it also removed almost 36 percent of the known Tor packets which resulted in an adjusted FNR on the first test. In the second best performing test, error rates were roughly 20 percent for both Tor and non-Tor traffic. Although the observed FNR and FPR were high, use of this type of analysis on a network in real-time could result in a reduction of Tor traffic by approximately 80 percent—significantly reducing the overall threat level.

V. CONCLUSION

Obfuscation technologies have evolved into a vast network capable of providing on-demand obfuscation services world-wide. Further, the anonymity provided by obfuscation technologies, such as Tor, allows users to disassociate themselves from their online activity. While this can be used for legitimate purposes, it can also be used to mask the origin of malicious activity. As a result, the amount of obfuscated traffic, malicious or otherwise, on private and DOD networks is unknown. While there have been numerous studies conducted to identify new ways to detect obfuscated traffic, this study focused on detecting the popular obfuscation service Tor using a simple approach. Using simple network traffic analysis techniques to detect and discern Tor from non-Tor traffic extends the compatibility of this study's approach with any network or server capable of monitoring incoming traffic. By implementing the observations in this study, the threat to DOD networks from Tor traffic can be reduced by at least 80 percent.[44]

A. RESULTS

This study used multiple clients with different operating systems to generate two sets of network traffic—Tor and non-Tor. Each set of traffic was analyzed using R-script to identify unique traits or characteristics. After analysis, each trait was tested to determine its ability to detect Tor traffic in a mixed dataset; this resulted in a 53 to 62.1 percent probability of correctly classifying Tor traffic. To improve the single-attribute probability, this study used a Naive Bayes multi-attribute prediction model on the same mixed dataset. To test the ability of each unique characteristic to pre-filter the mixed dataset, three separate filters were created. The filters were used to reduce the number of IP packets that did not exhibit the typical Tor characteristics observed in the experiment's dataset. The best results were observed using only IP TTL characteristics to filter the dataset. After filtering and analysis, IP TTL filtering resulted in a true-positive rate (TPR)

[44] Naive Bayes analysis of an IP TTL filtered training set and an unfiltered test set resulted in a TPR of approximately 80 percent for Tor traffic. Because the analysis was run on an unfiltered test set, no known Tor packets were filtered before analysis.

of 91.3 percent for Tor traffic and a false-positive rate (FPR) of 7.2 percent for non-Tor traffic.

While filtering the dataset improved the TPR and FPR, it also inadvertently removed known Tor packets from the multi-attribute analysis. Adjusting for the known Tor packets removed by the IP TTL filter decreased the TPR from 91.3 to 58.5 percent. Though the adjusted TPR may appear prohibitively low, it could be improved by examining the network traffic by source IP rather than packet by packet. This approach could examine entire groups of packets and increase the probability of detection, as the overall group of activity from a single IP would be less affected by filtering.

B. FUTURE RESEARCH

Based on the observations in this study, Tor traffic likely exhibits additional unique characteristics that could be identified using network traffic analysis. The study focused on data and traits that could be derived with little to no additional calculations other than the packet information logged in Snort. However, by examining the data for characteristics exhibited over multiple packets or an entire session, additional indicators should be identified for use in a multi-attribute analysis model. This study proposes three additional indicators of Tor traffic as well as a way to process and detect that activity in real-time.

1. High RTT

Based on Tor's geographically dispersed architecture and routing schema, this research hypothesized that the same linear correlation between geographic distance and round-trip time (RTT), as observed by Claffy et al. [31], would be present in Tor traffic. While this study observed that a majority of known Tor exit nodes use Linux-based operating systems, it is also possible the IP TTL values of Tor traffic are lower overall because of the additional hops required by the active Tor circuit. However, leveraging the IP TTL observations of this study, the high RTT could be used as an additional discriminator in multi-attribute analysis.

2. HTTP Flow Analysis

Based on observations by Barker et al. [4], it is possible to identify Tor traffic streams using the packet length of HTTP flow packets three and five. However, this research proposes there is a sufficient separation between the sizes for flow packet three; this separation could be used as an additional discriminator in multi-attribute analysis. To verify the results of [4], Tor and non-Tor HTTP flow packet sizes could be compared to determine whether such a separation exists.

3. Varied Source IP

Building on work by Kelly [9], it may be possible to identify Tor traffic by monitoring for a IP address change from an authenticated user during a session. According to Syverson [5], an active Tor session constructs a new circuit when the 10-minute maxcircuitdirtiness time expires.[45] This construction should result in a different exit node, which the server views as a change to the client's IP address. If a client has an exit node that continually shifts every 10 minutes, that may indicate the use of an active Tor circuit. If proven, varying source IP could be used in as a discriminator in multi-attribute analysis.

4. Vector Relational Data Modeling

To facilitate real-time detection, this study proposes use of a vector relational data model (VRDM) approach. Proof-of-concept experiments by Kelly [6] and Seng [45] were successful at detecting malicious activity from users by examining their overall behavior rather than through packet-by-packet analysis. Both of these studies used a VRDM multi-criteria decision model with pre-defined sets of indicators, or *metarules*, to detect malicious activity.[46] To ensure non-malicious traffic was not incorrectly marked because it met the requirement of a metarule, both studies used an accumulator as a central

[45] By default, each Tor circuit is used for a maximum of 10 minutes; however, users have the option of creating a new circuit when desired [5].

[46] According to Seng, metarules are not dependent on data patterns but use relationships among conceptual components to deduce meaning. For example, BBNM metarules are: "If an Actor is bad, then all its Sessions are bad," "If a Session is bad, then all its Behaviors are highly threatening," and "If a Session is bad, then the Actor is bad." In short, metarules are used to infer intent similarly in the way a human analyst would think [45].

decision engine capable of taking inputs from each metarule. The proposed accumulator model is provided in Appendix E.

5. Real-Time Detection Axioms

Detecting Tor traffic using network traffic analysis in real-time will require a different evaluation and detection approach. This study proposes the use of detection axioms to simplify information flow and programming in a real-time detection model. Each axiom should be capable of breaking down the basic input/output requirements and decision points during each step. Proposed axioms for the common Tor characteristics examined during this study as well as future work recommendations are provided in Appendix E.

C. FINAL THOUGHTS

From start to finish, this study was intellectually stimulating and required continual, in-depth research into multiple disciplines. Construction of the testing infrastructure proved more complex due to interoperability and communication requirements within the system. Integrating multiple software suites from different manufactures required additional considerations. After the final datasets were generated, analysis with R-script seemed a logical next step. While R provided a robust data analysis toolset, it required countless hours of research and testing to achieve a beginner level of proficiency. Developing a functioning R-script capable of accurate and efficient evaluation required multiple iterations of testing, tuning, and retesting. In the end, the virtual lab, and analysis with R-script provided a concrete foundation to test the theories put forward by this body of research.

APPENDIX A. MANUAL FIREFOX PROXY CONFIGURATION

There are two main steps required to configure Firefox to use an active Tor circuit. First you must have the Tor browser open to establish the circuit, and second you must manually enter the required manual proxy data into Firefox. To access the required proxy settings, open Firefox, go to Open Menu, next select "Options," then "Advanced," select the "Network" tab, click on "Settings," and select "Manual proxy configuration" radio button. Next, enter "127.0.0.1" into the SOCKS Host field, and "9150" into the port field. Lastly, select the "Remote DNS" radio button if it has not already been selected. Figure 26 illustrates the manual configuration.

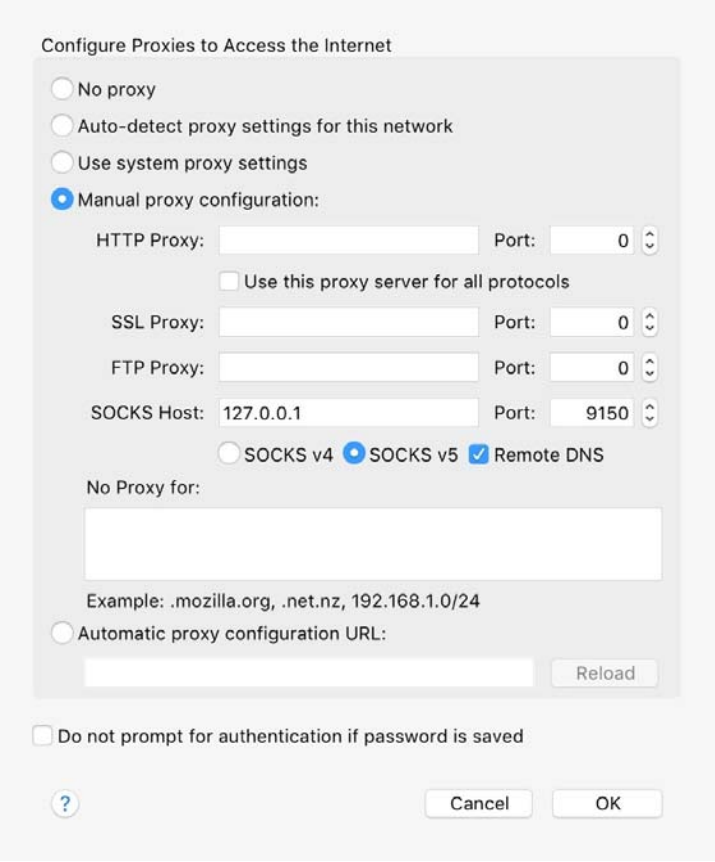


Figure 26. Required Firefox manual proxy configuration settings to proxy over an active Tor circuit.

Once completed the manual proxy settings have been completed, navigate to check.torproject.org to verify whether Firefox is using an active Tor circuit (see Figure 27).[47] While this configuration works, a drawback is that the Tor browser must be kept open to maintain the connection to the Tor network; otherwise, "no proxy" must be selected in the Firefox advanced settings to access the Internet.

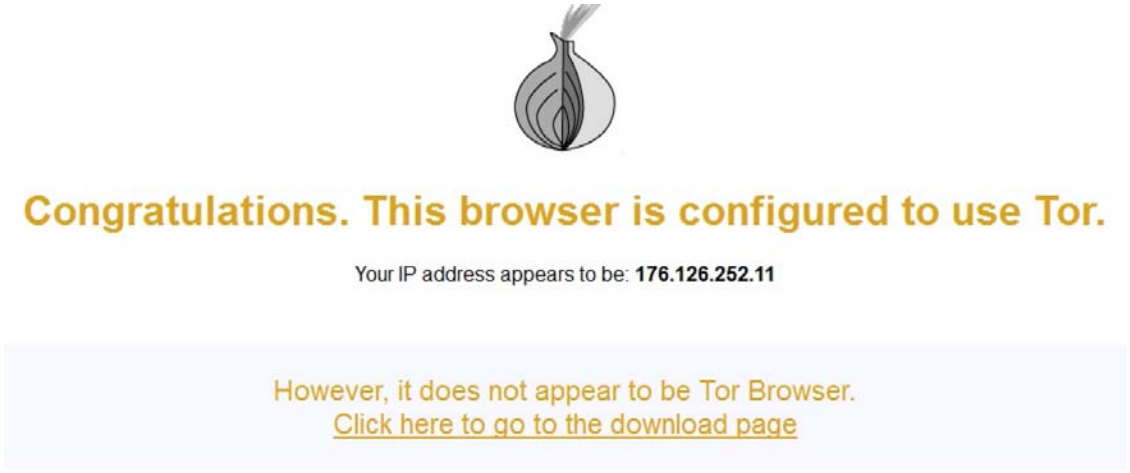


Figure 27. Successful Firefox manual proxy over Tor.

[47] http://www.wikihow.com/Use-Tor-With-Firefox provided step by step directions to manually configure Firefox to use an active Tor circuit. Accessed 15 November 2016.

APPENDIX B. SNORT AND LOGPARSER 2.2 CONFIGURATION

A. SNORT

Following configuration steps by [6], Snort 2.9.2.2 was installed and configured to operate in IDS mode on the DarkNet web server. We used a windows batch script to run Snort as a packet sniffer and modified the snort.conf file to log all information into the local MSSQL server for storage and analysis. To successfully log all packet information to MSSQL, Snort natively provides the necessary SQL code to build the required tables; this code is located in c:\snort\schemas\create_mssql.sql.

1. MSSQL Logging

Eight steps are required to configure Snort to log all information in MSSQL. As detailed by [6], the eight steps are as follows:

1. Open a "new query" window in MSSQL Sever Management Studio and enter the following lines:
2. Create database snort_db
3. Go
4. Use snort_db
5. Go
6. Copy and paste the contents of the create_mssql.sql file into the query window.
7. Execute
8. Grant all database permissions to the snort user.

2. snort.conf Configuration

After the script successfully generates the required tables in MSSQL, enable logging by entering the following into the snort.conf:

```
output log_tcpdump: /Snort/log/tcpdump.log
output database: log, mssql, user=snort password=snort
dbname=SNORT host=DARKNETGINA\DARKNETSQL encoding=ascii
log IP any any -> any any
```

3. Windows Batch File Configuration

Once the snort.conf file is updated, create a windows batch file to run Snort in IDS mode in the background while executing the custom snort.conf file. Once manually executed, the batch file opens a windows command shell and prints the verbose details of all packets observed. The batch file we created, snort.bat, contains the following line and needs to be manually started:

c:\Snort\bin\snort -i 1 -vd -c c:\snort\etc\snort.conf

B. LOGPARSER 2.2

On the DarkNet web server, W3C logging was used to capture client-specific information not logged by Snort. The problem, however, is that W3C records all logs in text format, but IIS is not configured to automatically write these logs to MSSQL. Microsoft LogParser 2.2 was used to bridge the gap between the W3C logs and MSSQL. LogParser is capable of moving the logs from their native text format into MSSQL (or many other formats). However, it does have limitations. It must be manually executed each time the logs are parsed, and it will only parse data that is both defined in the script and present in the logs. The latter is problematic since Microsoft IIS and W3C logging do not write a globally unique identification (GUID) field to the logs. It is possible, however, to write in a placeholder that allows the LogParser to write the data from the logs and permit MSSQL to assign a searchable identity to each row. Two scripts are required to properly use LogParser in our configuration. The first is designed to create the database and executed only once. The second is used to periodically run the parser.

1. LogParser 2.2 MSSQL Database Creation

The first script is designed to create a database in MSSQL called “ParsedIISLogs” and subsequently parse any log data at the same time. Specifically, the script parses the time, server IP addresses, server ports, client IP addresses, authenticated usernames, client user-agent strings, and the time taken by the web server to complete a request. The following script only needs to be run once to create the table:

logparser -i:IISW3C -o:SQL -server:DARKNETGINA\DARKNETSQL -database:DARKNETIIS -driver:"SQL Server" -username:<username> -password:<password> -createTable:ON "SELECT date AS Date, time AS Time, s-ip AS ServerIP, s-port AS ServerPort, cs-username AS Username, c-ip AS Client, cs(User-Agent) AS UserAgent, time-taken as TimeTaken INTO ParsedIISLog FROM C:\inetpub\logs\LogFiles\W3SVC1001830619\u_ex*.log"

2. LogParser 2.2 On-Demand Parsing

After the first LogParser script successfully executes, the second script pulls new data from the defined log(s) and writes them to the "ParsedIISLogs" database. To ensure only new data is written to MSSQL when executed, the following script creates a checkpoint where the last data was imported from; the script is run manually, as needed.

logparser -i:IISW3C -o:SQL -server:DARKNETGINA\DARKNETSQL -database:DARKNETIIS -driver:"SQL Server" -username:<username> -password:<password> -ignoreIdCols:on -icheckpoint:MyCheckpoint.lpc "SELECT date AS Date, time AS Time, s-ip AS ServerIP, s-port AS ServerPort, cs-username AS Username, c-ip AS Client, cs(User-Agent) AS UserAgent, time-taken AS TimeTaken INTO ParsedIISLog FROM C:\inetpub\logs\LogFiles\W3SVC1001830619\u_ex*.log"

Although not visible in the on-demand parsing script, the table created by the first script has to be modified to allow MSSQL to add a GUID field after the data has been parsed into the database. What is included in the on-demand parsing script is the "-ignoreIdCols:on" option. This tells the log parser to write to the defined columns and ignore any other columns as long as they are configured as *identity* elements. To configure the identity element and add the GUID field without errors, the identity element must be manually added to the MSSQL table.[48] The identity element allows data from the IIS logs to be parsed while ignoring the additional unique identification data field not present in the on-demand parsing script. The identity field requires two defined characteristics to run properly. First, it must be configured with "int" as the datatype, and second, it needs to be defined as an identity element (see Figure 28).

[48] Information used to configure LogParser 2.2 and identity rows can be accessed at https://www.simple-talk.com/sql/sql-tools/microsofts-log-parser-utility-swell-etl/. Accessed 18 November 2016.

```
USE [DarkNetIIS]
GO

/****** Object:  Table [dbo].[ParsedIISLog]    Script Date: 11/22/2016 10:06:09 PM ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

SET ANSI_PADDING ON
GO

CREATE TABLE [dbo].[ParsedIISLog](
    [Date] [datetime] NULL,
    [Time] [datetime] NULL,
    [ServerIP] [varchar](255) NULL,
    [ServerPort] [int] NULL,
    [Username] [varchar](255) NULL,
    [Client] [varchar](255) NULL,
    [UserAgent] [varchar](255) NULL,
    [TimeTaken] [int] NULL,
    [IISIdentity] [int] IDENTITY(1,1) NOT NULL
) ON [PRIMARY]

GO

SET ANSI_PADDING OFF
GO
```

Figure 28. Table configuration required to allow MSSQL GUID assignment after data is Parsed by LogParser 2.2.

APPENDIX C. NAIVE BAYES MULTI-ATTRIBUTE TESTING RESULTS

A total of nine Naive Bayes multi-attribute tests were conducted on both filtered and unfiltered datasets. Each dataset was filtered using the same criteria as the test set during each test. Testing consisted of three main tests focused on evaluating the dataset based on a set of pre-filters. The first test, examined Naive Bayes performance filtering the dataset using common Tor IP TTL, packet size, and TCP offset values. The second test analyzed a dataset filtered to include only packets exhibiting common Tor IP TTL and TCP offset, and the third test examined only packets exhibiting common Tor IP TTL values. The three main tests examined the performance of their respective filters on three separate sub-tests. First, on a filtered training and test set, next on a filtered training and unfiltered test set, and finally on an unfiltered training and filtered test set. Of note, the calculated FNR for Tor is a result of the Naive Bayes classification only and is not adjusted for the amount of Tor packets removed from the dataset by filtering.

A. MULTI-ATTRIBUTE TEST ONE

Multi-attribute test one examined the performance of the Naive Bayes classifier using three filters. Filter one selected only packets which exhibited a TTL between 43 and 60, with the exception of packet sizes 52, 56, or 57; filter two selected only packets that exhibited a TCP offset value of 10, 11, 12, 14, 15; and filter three selected only packet sizes 41, 52, 64, 127, 254, 713, or 1500. Tables 12–14 provide a summary of the results.

Table 12. Naive Bayes test 1A filtered training and test results.

Filtered Training and Filtered Test	Tor	Non-Tor
Packets Classified Correctly	29954	41991
Packets Misclassified	47347	15205
False Positive Rate		26.6%
False Negative Rate	61.3%	

Table 13. Naive Bayes test 1B filtered training and unfiltered test results.

Filtered Training and Unfiltered Test	Tor	Non-Tor
Packets Classified Correctly	39384	42500
Packets Misclassified	48412	41296
False Positive Rate		49.3%
False Negative Rate	55.1%	

Table 14. Naive Bayes test 1C unfiltered training and filtered test results.

Unfiltered Training and Filtered Test	Tor	Non-Tor
Packets Classified Correctly	74645	7728
Packets Misclassified	2656	49468
False Positive Rate		86.5%
False Negative Rate	3.4%	

B. MULTI-ATTRIBUTE TEST TWO

Multi-attribute test two examined the performance of the Naive Bayes classifier using two filters. Filter one selected only packets which exhibited a TTL between 43 and 60, with the exception of packet sizes 52, 56, or 57; filter two selected only packets which exhibited a TCP offset value of 10, 11, 12, 14, 15. Tables 15–17 provide a summary of the results.

Table 15. Naive Bayes test 2A filtered training and test results.

Filtered Training and Filtered Test	Tor	Non-Tor
Packets Classified Correctly	3595	58090
Packets Misclassified	39313	1437
False Positive Rate		2.4%
False Negative Rate	91.6%	

Table 16. Naive Bayes test 2B filtered training and unfiltered test results.

Filtered Training and Unfiltered Test	Tor	Non-Tor
Packets Classified Correctly	43664	77314
Packets Misclassified	40132	10482
False Positive Rate		11.9%
False Negative Rate	47.9%	

Table 17. Naive Bayes test 2C unfiltered training and filtered test results.

Unfiltered Training and Filtered Test	Tor	Non-Tor
Packets Classified Correctly	4449	57188
Packets Misclassified	38459	2339
False Positive Rate		3.9%
False Negative Rate	89.6%	

C. MULTI-ATTRIBUTE TEST THREE

Multi-attribute test three examined the performance of the Naive Bayes classifier using one filters. Filter one selected only packets which exhibited a TTL between 43 and 60, with the exception of packet sizes 52, 56, or 57. Tables 18–20 provide a summary of the results.

Table 18. Naive Bayes test 3A filtered training and test results.

Filtered Training and Filtered Test	Tor	Non-Tor
Packets Classified Correctly	51,353	36,343
Packets Misclassified	4,885	2,807
False Positive Rate		7.2%
False Negative Rate	8.7%	

Table 19. Naive Bayes test 3B filtered training and unfiltered test results.

Filtered Training and Unfiltered Test	Tor	Non-Tor
Packets Classified Correctly	70218	66211
Packets Misclassified	17578	17585
False Positive Rate		21.0%
False Negative Rate	20.0%	

Table 20. Naive Bayes test 3C unfiltered training and filtered test results.

Unfiltered Training and Filtered Test	Tor	Non-Tor
Packets Classified Correctly	2339	2211
Packets Misclassified	53899	36939
False Positive Rate		94.4%
False Negative Rate	95.8%	

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX D. DATA ANALYSIS R-SCRIPT

R was used to analyze each key cyber indicator to identify distinct characteristics and to examine the data for correlations between attributes. The goal of analysis was to calculate the probability of misclassification when each trait was used by itself to identify Tor traffic in the larger, combined, dataset. A total of six single attribute tests, ten multi-attribute tests were conducted using the following R-script:

```
#Load Required Libraries
library(class)
library(e1071)
library(dplyr)
library(ggplot2)
library(visreg)
library(boot)
library(psych)

########Load Required Data
known_tor <- read.csv("C:/Users/trac/Desktop/DarkNet Testing/Log Data/DarkNet
                      Testing Logs/tor_mixset.csv," header = TRUE, as.is =
                      TRUE)

known_non_tor <- read.csv("C:/Users/trac/Desktop/DarkNet Testing/Log Data/DarkNet
                      Testing Logs/nontor_mixset.csv," header = TRUE, as.is =
                      TRUE)

train_mixset <- read.csv("C:/Users/trac/Desktop/DarkNet Testing/Log Data/DarkNet
                      Testing Logs/comb_snort_mixset_lean2_75train.csv,"
                      header = TRUE, as.is = TRUE)

test_mixset <- read.csv("C:/Users/trac/Desktop/DarkNet Testing/Log Data/DarkNet
                      Testing Logs/comb_snort_mixset_lean2_25test.csv,"
                      header = TRUE, as.is = TRUE)

exit_node <- read.csv("C:/Users/trac/Desktop/DarkNet Testing/Log Data/DarkNet
                      Testing Logs/Exit Nodes/Known_exit_node.csv," header =
                      TRUE, as.is = TRUE)

########Create the mixed dataset with Tor column
known_non_tor$tor <- 0
known_tor$tor <- 1
mixset <- rbind(known_non_tor, known_tor)
```

########Generate Data Histograms based on relationships between the attributes

#Full mixed correlation dataset trimmed to include only TTL, IP Length, TCP offset, and tor columns.

```
full_correlation <- mixset[,c(12,16,24,30)]
```

#Tor only correlation dataset trimmed to include only TTL, IP Length, TCP offset, and tor columns

```
tor_correlation <- known_tor
tor_correlation$tor <- 1
tor_correlation <- tor_correlation[,c(12,16,24,30)]
```

#Attribute correlation on full unfiltered dataset

```
pairs.panels(full_correlation)

pairs(full_correlation[1:4], main = "Mixset Trait Correlation—No Filter,"
pch = 21, bg = c('red','blue', 'green')[unclass(tor_correlation$tor)])
```

#Unfiltered known Tor dataset

```
pairs.panels(tor_correlation[1:3])

pairs(tor_correlation[1:4], main = "Known Tor Packet Trait Correlation—No Filter,"
pch = 21, bg = c('red','blue', 'green')[unclass(tor_correlation$tor)])
```

#full dataset with only packets exhibiting Tor TTL, TCP offset and Packet size attributes

```
tor_correlation_all_obv_trait <- tor_correlation %>% filter(ip_ttl > 43 & ip_ttl < 59 &
                                 ip_ttl != 52 & ip_ttl != 56 & ip_ttl != 57 | _off == 10 |
                                 tcp_off == 11 | tcp_off == 12 | tcp_off == 14 | tcp_off ==
                                 15 | ip_len == 52 | ip_len == 1500| ip_len == 41 | ip_len ==
                                 64 | ip_len == 127 | ip_len == 254 | ip_len == 713)

pairs.panels(tor_correlation_all_obv_trait[1:3])

pairs(tor_correlation_all_obv_trait[1:4], main = "Known Tor Packet Trait Correlation—
                                 All Filters," pch = 21, bg = c('red','blue',
                                 'green')[unclass(tor_correlation_all_obv_trait$tor)])
```

#full dataset with only packets exhibiting Tor TTL and TCP offset attributes

```
tor_correlation_ttl_tcp <- tor_correlation %>% filter(ip_ttl > 43 & ip_ttl < 59 & ip_ttl !=
                           52 & ip_ttl != 56 & ip_ttl != 57 | tcp_off == 10 | tcp_off ==
                           11 | tcp_off == 12 | tcp_off == 14 | tcp_off == 15)

pairs.panels(tor_correlation_ttl_tcp[1:3])
```

```
pairs(tor_correlation_ttl_tcp[1:4], main = "Known Tor Packet Trait Correlation—TTL/
                                    OFF Filter,"pch = 21, bg = c('red','blue',
                                    'green')[unclass(tor_correlation_ttl_tcp$tor)])
```

#Histograms for full dataset with only packets exhibiting Tor ttl attribute

```
tor_correlation_ttl <- tor_correlation %>% filter(ip_ttl > 43 & ip_ttl < 59 & ip_ttl != 52
                                    & ip_ttl != 56 & ip_ttl != 57)

pairs.panels(tor_correlation_ttl[1:3])

pairs(tor_correlation_ttl[1:4], main = "Known Tor Packet Trait Correlation—TTL Filter,"
                                    pch = 21, bg = c('red','blue',
                                    'green')[unclass(tor_correlation_ttl$tor)])
```

########Start Single-Attribute Testing

####KCC One: Low TTL Count

##Filtered—Measure the linear regression of Tor TTL count. Results will provide probability traffic is tor based on TTL value

```
mixset2 <- mixset %>% filter(ip_ttl > 43 & ip_ttl < 59 & ip_ttl != 52 & ip_ttl != 56 &
                                    ip_ttl != 57)

model1 <- glm(tor ~ ip_ttl, data = mixset2, family='binomial')
summary(model1)
visreg(model1, 'ip_ttl', scale='response', xlab = 'TTL Value', ylab = 'Probability', main
                                    = 'Tor Classification Probability based on Filtered TTL')
```

#Calculate misclassification rate for TTL filtered dataset

```
test <- glm(tor ~ ip_ttl, mixset2, family = "binomial")
pi.hat <- round(predict(test,type="response"),3)
y <- mixset2$tor
table(y,pi.hat>5)
mean(y != (pi.hat>.5))
```

#Cross Validation of misclassification rate for TTL filtered dataset

```
cost <- function(y,pi.hat){mean(y!=(pi.hat>.5))}
cv.glm(mixset2,test,cost(y,pi.hat),K=10)$delta
```

##Unfiltered—Measure the linear regression of Tor TTL count. Results will provide probability traffic is tor based on TTL value

```
model1 <- glm(tor ~ ip_ttl, data = mixset, family='binomial')
summary(model1)
```

```
visreg(model1, 'ip_ttl', scale='response', xlab = 'TTL Value', ylab = 'Probability', main
                                = 'Tor Classification Probability based on Unfiltered TTL')
```

#Calculate misclassification rate for TTL unfiltered dataset

```
test <- glm(tor ~ ip_ttl, mixset, family = "binomial")
pi.hat <- round(predict(test,type="response"),3)
y <- mixset$tor
table(y,pi.hat>5)
mean(y != (pi.hat>.5))
```

#Cross Validation of misclassification rate for TTL unfiltered dataset

```
cost <- function(y,pi.hat){mean(y!=(pi.hat>.5))}
cv.glm(mixset,test,cost(y,pi.hat),K=10)$delta
```

####KCC Two: Common IP Packet Size

##Filtered—Measure the linear regression of common Tor packet size. Results will provide probability traffic is tor based on filtered IP packet size

```
mixset2 <- mixset %>% filter(ip_len == 52 | ip_len == 1500 | ip_len == 41 | ip_len == 64
                                | ip_len == 127 | ip_len == 254 | ip_len == 713)

model1 <- glm(tor ~ ip_len, data = mixset2, family='binomial')
summary(model1)
visreg(model1, 'ip_len', scale='response', xlab = 'IP Packet Length', ylab =
                                'Probability', main = 'Tor Classification Probability based
                                on Filtered IP Packet Length')
```

#Calculate misclassification rate for filtered common IP packet dataset

```
test <- glm(tor ~ ip_len, mixset2, family = "binomial")
pi.hat <- round(predict(test,type="response"),3)
y <- mixset2$tor
table(y,pi.hat>5)
mean(y != (pi.hat>.5))
```

#Cross validation of misclassification rate for filtered common IP packet dataset

```
cost <- function(y,pi.hat){mean(y!=(pi.hat>.5))}
cv.glm(mixset2,test,cost(y,pi.hat),K=10)$delta
```

##Unfiltered—Measure the linear regression of common Tor packet size. Results will provide probability traffic is Tor based on unfiltered IP packet size

```
model1 <- glm(tor ~ ip_len, data = mixset, family='binomial')
summary(model1)
```

```
visreg(model1, 'ip_len', scale='response', xlab = 'IP Packet Length', ylab =
                        'Probability', main = 'Tor Classification Probability based
                        on IP Packet Length')
```

#Calculate misclassification rate for unfiltered Common IP packet dataset

```
test <- glm(tor ~ ip_len, mixset, family = "binomial")
pi.hat <- round(predict(test,type="response"),3)
y <- mixset$tor
table(y,pi.hat>5)
mean(y != (pi.hat>.5))
```

#Cross Validation of misclassification rate for unfiltered Common IP packet dataset

```
cost <- function(y,pi.hat){mean(y!=(pi.hat>.5))}
cv.glm(mixset,test,cost(y,pi.hat),K=10)$delta
```

####KCC Three: Common TCP Offset

##Filtered—Measure the linear regression of Tor TCP Offset. Results will provide probability traffic is tor based on filtered TCP Offset

```
mixset2 <- mixset %>% filter(tcp_off == 10 | tcp_off == 11 | tcp_off == 12 | tcp_off ==
                        14 | tcp_off == 15)
model1 <- glm(tor ~ tcp_off, data = mixset2, family='binomial')
summary(model1)
visreg(model1, 'tcp_off', scale='response', xlab = 'TCP Offset', ylab = 'Probability',
                        main = 'Tor Classification based on Filtered TCP Offset')
```

#Calculate misclassification rate for filtered TCP Offset dataset

```
test <- glm(tor ~ tcp_off, mixset2, family = "binomial")
pi.hat <- round(predict(test,type="response"),3)
y <- mixset2$tor
table(y,pi.hat>5)
mean(y != (pi.hat>.5))
```

#Cross validation of misclassification rate for filtered TCP Offset dataset

```
cost <- function(y,pi.hat){mean(y!=(pi.hat>.5))}
cv.glm(mixset2,test,cost(y,pi.hat),K=10)$delta
```

##Unfiltered—Measure the linear regression of Tor TCP Offset. Results will provide probability traffic is tor based on unfiltered TCP Offset

```
model1 <- glm(tor ~ tcp_off, data = mixset, family='binomial')
summary(model1)
visreg(model1, 'tcp_off', scale='response', xlab = 'TCP Offset', ylab = 'Probability',
                        main = 'Tor Classification based on Unfiltered TCP
                        Offset')
```

#Calculate misclassification rate for unfiltered TCP Offset dataset

```
test <- glm(tor ~ tcp_off, mixset, family = "binomial")
pi.hat <- round(predict(test,type="response"),3)
y <- mixset$tor
table(y,pi.hat>5)
mean(y != (pi.hat>.5))
```

#Cross Validation of misclassification rate for unfiltered TCP Offset dataset

```
cost <- function(y,pi.hat){mean(y!=(pi.hat>.5))}
cv.glm(mixset,test,cost(y,pi.hat),K=10)$delta
```

########Start Multi-Attribute Naive Bayes Testing

####Baseline Performance Test—Unfiltered training and unfiltered testing

#Naive Bayes analysis and prediction model based on unfiltered dataset

```
classifier_full <- naiveBayes(train_mixset[,2:4], factor(train_mixset[,5]) )
```

#Test Classifier against itself

```
table(predict(classifier_full, train_mixset[,-5]), factor(train_mixset[,5]) )
```

#Test classifier against unfiltered test dataset

```
table(predict(classifier_full, test_mixset[,-5]), factor(test_mixset[,5]) )
```

#########Test 1—Multi-Attribute Naive Testing using TTL, TCP offset, and IP packet size

####Filters for all three observed Tor characteristics

```
sample_data <- train_mixset %>% filter(ip_ttl > 43 & ip_ttl < 59 & ip_ttl != 52 & ip_ttl
                                != 56 & ip_ttl != 57 | tcp_off == 10 | tcp_off == 11 | tcp_off
                                == 12 | tcp_off == 14 | tcp_off == 15 | ip_len == 52 | ip_len
                                == 1500 | ip_len == 41 | ip_len == 64 | ip_len == 127 |
                                ip_len == 254 | ip_len == 713)

sample_data1 <- test_mixset %>% filter(ip_ttl > 43 & ip_ttl < 59 & ip_ttl != 52 & ip_ttl
                                != 56 & ip_ttl != 57 | tcp_off == 10 | tcp_off == 11 | tcp_off
                                == 12 | tcp_off == 14 | tcp_off == 15 | ip_len == 52 | ip_len
                                == 1500 | ip_len == 41 | ip_len == 64 | ip_len == 127 |
                                ip_len == 254 | ip_len == 713)
```

####Test 1A—Filtered training set and filtered test set

#Naive Bayes analysis and prediction model based on filtered dataset

```
classifier_filtered <- naiveBayes(sample_data[,2:4], factor(sample_data[,5]) )
```

#Test the training set against itself
table(predict(classifier_filtered, sample_data[,-5]), factor(sample_data[,5]))

#tests the test set against the training set
table(predict(classifier_filtered, sample_data1[,-5]), factor(sample_data1[,5]))

####Test 1B—Filtered training set and unfiltered test set

#Test classifier against unfiltered test dataset
table(predict(classifier_filtered, test_mixset[,-5]), factor(test_mixset[,5]))

####Test 1C—Unfiltered training set and filtered test set

#Test classifier against filtered test dataset
table(predict(classifier_full, sample_data1[,-5]), factor(sample_data1[,5]))

#########Test 2—Multi-Attribute Testing using only IP TTL and TCP offset

####Filters for two observed Tor characteristics
sample_data <- train_mixset %>% filter(ip_ttl > 43 & ip_ttl < 59 & ip_ttl != 52 & ip_ttl
!= 56 & ip_ttl != 57 | tcp_off == 10 | tcp_off == 11 | tcp_off
== 12 | tcp_off == 14 | tcp_off == 15)

sample_data1 <- test_mixset %>% filter(ip_ttl > 43 & ip_ttl < 59 & ip_ttl != 52 & ip_ttl
!= 56 & ip_ttl != 57 | tcp_off == 10 | tcp_off == 11 | tcp_off
== 12 | tcp_off == 14 | tcp_off == 15)

####Test 2A—Filtered training set and filtered test set

#Naive Bayes analysis and prediction model based on filtered dataset
classifier_filtered <- naiveBayes(sample_data[,2:4], factor(sample_data[,5]))

#Test the training set against itself
table(predict(classifier_filtered, sample_data[,-5]), factor(sample_data[,5]))

#tests the test set against the training set
table(predict(classifier_filtered, sample_data1[,-5]), factor(sample_data1[,5]))

####Test 2B—Filtered training set and unfiltered test set

#Test classifier against unfiltered test dataset
table(predict(classifier_filtered, test_mixset[,-5]), factor(test_mixset[,5]))

####Test 2C—Unfiltered training set and filtered test set

#Test classifier against filtered test dataset
table(predict(classifier_full, sample_data1[,-5]), factor(sample_data1[,5]))

#########Test 3—Multi-Attribute Naive Bayes Testing using only TTL Filter

####Filter for observed Tor TTL characteristics
sample_data <- train_mixset %>% filter(ip_ttl > 43 & ip_ttl < 59 & ip_ttl != 52 & ip_ttl != 56 & ip_ttl != 57)

sample_data1 <- test_mixset %>% filter(ip_ttl > 43 & ip_ttl < 59 & ip_ttl != 52 & ip_ttl != 56 & ip_ttl != 57)

####Test 3A—Filtered training set and filtered test set

#Naive Bayes model based on filtered dataset
classifier_filtered <- naiveBayes(sample_data[,2:4], factor(sample_data[,5]))

#Test the training set against itself
table(predict(classifier_filtered, sample_data[,-5]), factor(sample_data[,5]))

#tests the test set against the training set
table(predict(classifier_filtered, sample_data1[,-5]), factor(sample_data1[,5]))

#Verify amount of known tor exit nodes present in the filtered dataset
tor_exit_node <- exit_node$ip_src
tor_source_IP <- sample_data$ip_src

#Number of unique rows in the known Tor and exit node list
exit_row <- nrow(table(tor_exit_node)) #1106 unique entries
tor_row <- nrow(table(tor_source_IP)) #344 unique entires—There were
tor_row
tor_exit_intersect <- length(intersect(tor_exit_node,tor_source_IP))
tor_exit_intersect
tor_exit_intersect/tor_row

####Test 3B—Filtered training set and unfiltered test set

#Test classifier against unfiltered test dataset
table(predict(classifier_filtered, test_mixset[,-5]), factor(test_mixset[,5]))

####Test 3C—Unfiltered training set and filtered test set####

#Test classifier against filtered test dataset
table(predict(classifier_full, sample_data1[,-5]), factor(sample_data1[,5]))

APPENDIX E. PROPOSED AXIOMS AND ACCUMULATOR MODEL

A. DETECTION AXIOMS

The proposed axioms are written to detect both the characteristics observed during this study and the characteristics which may be observed in the proposed future research. Each axiom is explained as a step-by-step process from evaluation to detection and is illustrated using a simple input-output diagram. Within each axiom, the obfuscation threshold value (OTV) indicates a pre-defined rule which indicates a Tor characteristic has been observed, and the weighted obfuscation value (WOV) which signifies the weighted value that would be added to an accumulator.

1. IP TTL

Proposed three-step axiom to identify Tor traffic using IP time-to-live (TTL).

(1) Examine source port(tcp_sport); if greater than 1023,[49] examine source IP (ip_src) address.

(2) If ip_src is not a trusted IP address, examine TTL (ip_ttl) value in the IP header.

(3) If ip_ttl exceeds the OTV, mark as obfuscated and add the WOV to the accumulator.

Figure 29 provides a simplified illustration of the three-step process.

[49] As described by [33], ports 0–1023 are well-known ports and are normally assigned to specific applications or services. Registered and dynamic ports, 1024–49151 and 49152–65535 respectively, are unassigned and used by clients for communication.

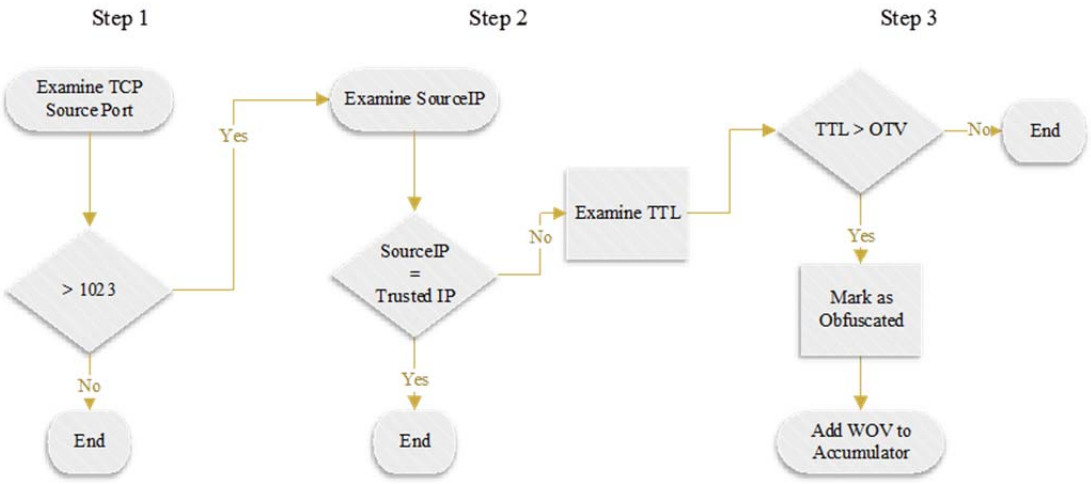


Figure 29. Proposed IP TTL three-step detection process.

2. IP Packet Length

Proposed five-step process to identify Tor traffic using IP packet length.

(1) k = (1500, 52, 638, 1150)

(2) Examine the IP length (ip_len) field in the IP header; if ip_len is an element of k, correlate the associated socket-pair using CID.

(3) Search back through the last 30 seconds for related packets using socket-pair and reverse socket-pair.

(4) Examine all packets' ip_len field, in aggregate, to check whether they exhibit common Tor packet sizes.

(5) If over 50 percent of the related packets exhibit common Tor packet sizes, mark as obfuscated and add the WOV to the accumulator.

Figure 30 provides a simplified illustration of the five-step process.

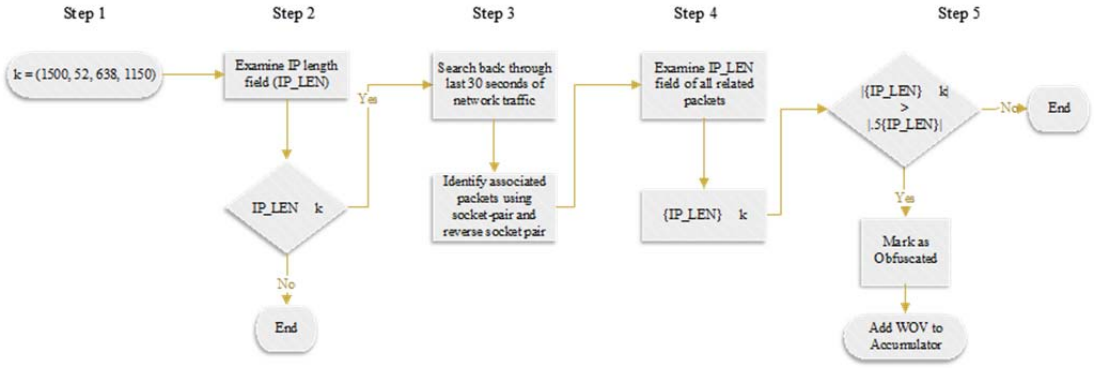


Figure 30. Proposed IP packet length five-step detection process.

3. TCP Offset

Proposed two-step process to identify Tor traffic using TCP offset value.

(1) Examine the TCP offset (tcp_off) field in the TCP header.

(2) If the tcp_off value exceeds the Tor TCP offset OTV, mark as obfuscated and add the WOV to the accumulator.

Figure 31 provides a simplified illustration of the two-step process.

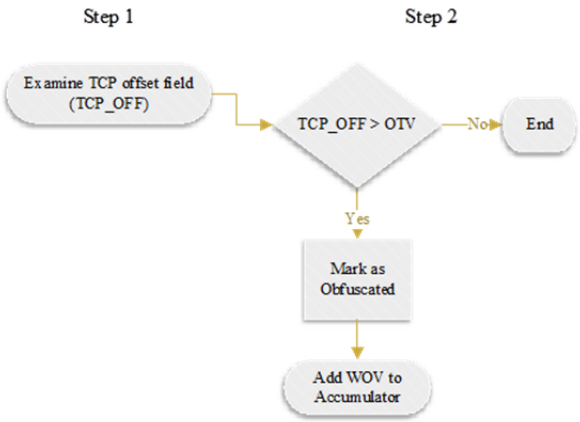


Figure 31. Proposed TCP offset two-step detection process.

4. Known Tor Exit Node

Proposed three-step process to identify obfuscated traffic using a blacklist of known Tor exit nodes.

(1) Examine the source IP field.

(2) If source IP is present on the Tor exit-node list, mark as obfuscated and add the max WOV to the accumulator.

(3) If source IP is not on Tor exit-node list after traffic has been marked as obfuscated by the decision model, add source IP to local bad actor list.

Figure 32 provides a simplified illustration of the three-step process.

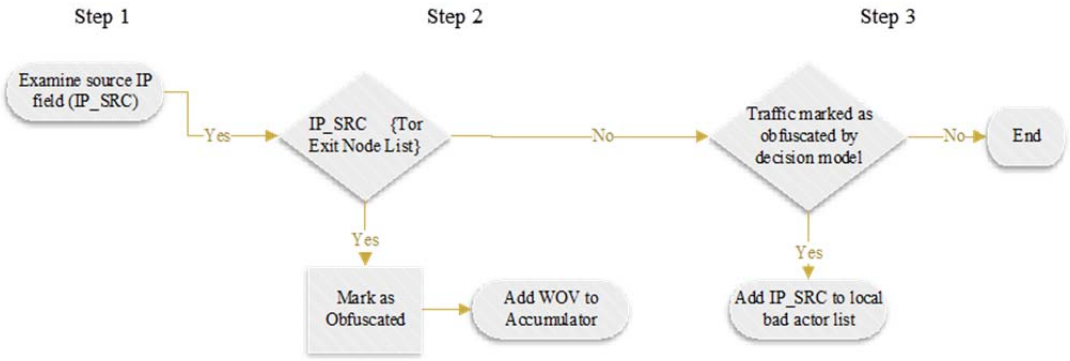


Figure 32. Proposed known Tor exit node three-step detection process.

5. High RTT

Proposed five-step process to identify Tor traffic using high RTT.

(1) To define a dynamic starting point, the TCP *syn-ack* handshake flag (18) is used to locate to defines a dynamic starting point from which to work.

(2) Once identified, the row CID (event ID) is used to locate the predictive acknowledgement number, and the socket-pair for the packet with the syn-ack flag set.

(3) Next, look back five seconds to locate the related *syn* packet by searching for the reversed socket-pair and the sequence number. The correct

sequence number is derived by subtracting one from the predictive acknowledgement number from the original syn-ack flagged packet.

(4) If an associated syn flagged packet is found, the CIDs are used to collect the timestamps for each packet, which measures the time difference or RTT for the two packets.

(5) Last, the observed RTT is compared to KCC Two's OTV to determine whether it is obfuscated. If the RTT exceeds the OTV, mark as obfuscated and add the WOV to the accumulator.

Figure 33 provides a simplified illustration of the five-step process.

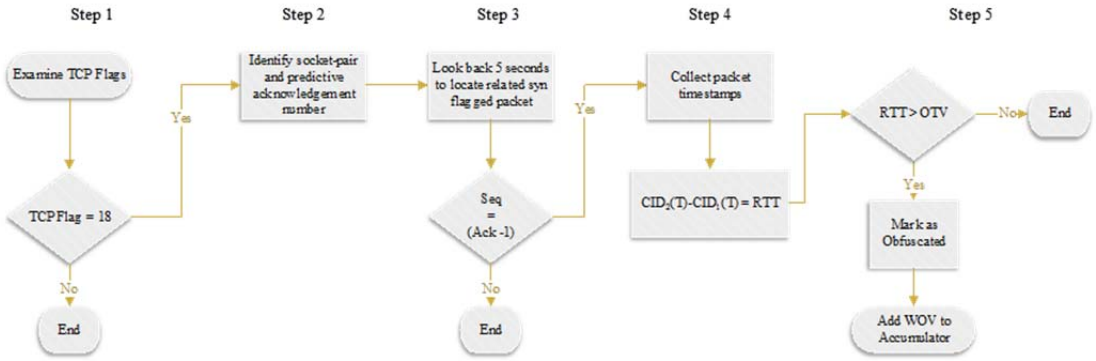


Figure 33. Proposed high RTT five-step detection process.

6. HTTP Flow Analysis

Proposed three-step process to identify Tor traffic using HTTP flow analysis.

(1) To define a dynamic starting point, the TCP *syn-ack* handshake flag (18) is used to locate to defines a dynamic starting point from which to work.

(2) Correlate the associated socket-pair for HTTP flow identification and monitor incoming traffic for five seconds to detect HTTP flow packet 3 using associated socket-pair.

(3) Examine IP length field. If ip_len field is between 131–152 bytes, mark as obfuscated and add the WOV to the accumulator.

Figure 34 provides a simplified illustration of the three-step process.

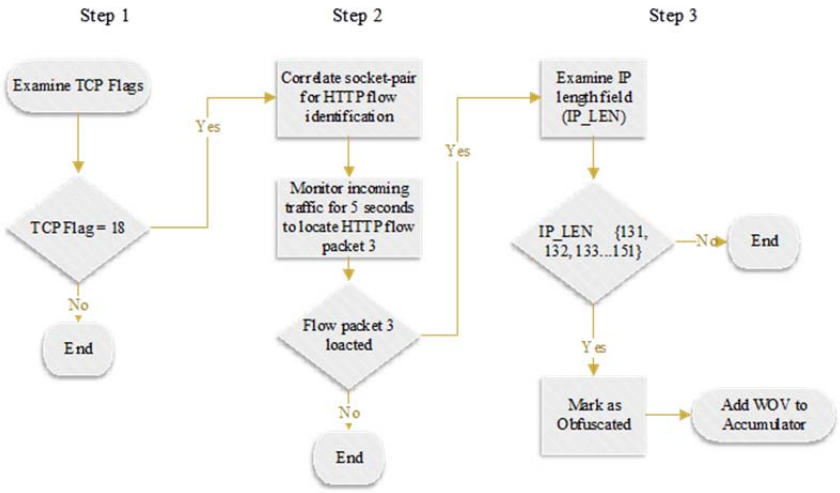


Figure 34. Proposed HTTP flow analysis three-step detection process.

7. Varied Source IP

Proposed three-step process to monitor for and identify Tor traffic using varying source IP after authentication.

(1) After HTTPS user authentication, correlate authenticated username and client IP (c-ip) pair.

(2) Every 60 seconds, look back 60 seconds of network traffic, and continue lookbacks for 15 60-second iterations.

(3) If a change in c-ip is detected, mark as obfuscated and add the WOV to the accumulator.

Figure 35 provides a simplified illustration of the proposed three-step process.

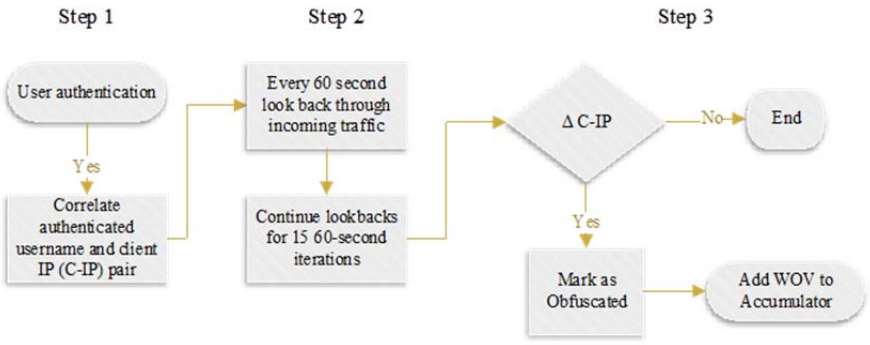
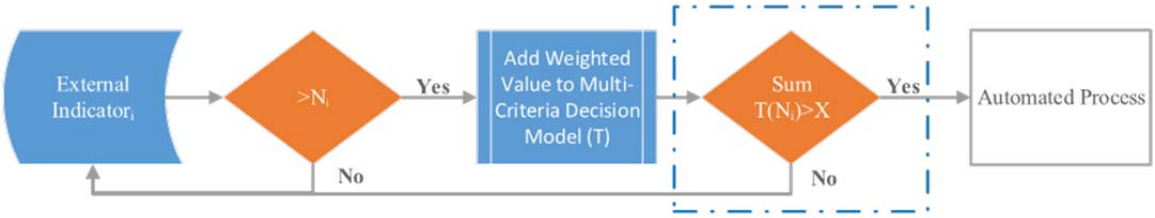


Figure 35. Proposed varied source IP three-step detection process.

B. ACCUMULATOR

Each indicator that is met during evaluation will provide input the accumulator. Specifically, the accumulator will iteratively take input from each indicator to determine whether users are attempting to obfuscate their activity. Then, after receiving input from all indicators, the accumulator can make an overall determination on whether the user is attempting to obfuscate their activity. Figure 36 provides an overview of the location and configuration for the proposed accumulator.



The accumulator appears within the dotted blue line.

Figure 36. Proposed multi-criteria accumulator model.

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

[1] L. Wang, K. P. Dyer, A. Akella, T. Ristenpart, and T. Shrimpton, "Seeing through network-protocol obfuscation," in *ACM Special Interest Group on Security, Audit and Control Conference Computer Commun. Sec.*, Denver, CO, 2015, pp. 57–69.

[2] S. Chakravarty, "Traffic analysis attacks and defenses in low latency anonymous communication," Ph.D. dissertation, Graduate School of Arts and Sciences, Columbia University, New York, 2014.

[3] D. L. Chaum, "Untraceable electronic mail, return addresses, and digital pseudonyms," *Communications of the ACM,* vol. 24, no. 2, pp. 84–88, Feb. 1981.

[4] J. Barker, P. Hannay, and P. Szewczyk, "Using traffic analysis to identify the second generation onion router," in *IEEE/IFIP International Conference on Embedded and Ubiquitous Comput.*, Melbourne, Australia, 2011, pp. 72–78.

[5] P. Syverson, "A peel of onion," *Annual Computer Security Applications Conference,* Orlando, FL, 2011, pp. 123–135.

[6] R. F. Kelly, "Automated cyber threat analysis and specified process using vector relational data modeling," M.S. Thesis, GSOIS, Naval Postgraduate School, Monterey, CA, 2014.

[7] D. Herrmann, R. Wendolsky, and H. Federrath, "website fingerprinting," in *ACM Workshop on Cloud Comput. Security*, Chicago, IL, 2009, pp. 1–13.

[8] H. Wang, C. Jin, and K. G. Shin, "Defense against spoofed IP traffic using hop-count filtering," *IEEE/ACM Trans.Netw.*, vol. 15, no. 1, pp. 40–53, Feb. 2007.

[9] R. F. Kelly, "Discerning anonymity," unpublished, Naval Postgraduate School, Monterey, CA.

[10] A. Biryukov, I. Pustogarov, and R. P. Weinmann, "Trawling for Tor hidden services: Detection, measurement, deanonymization," *Proc IEEE Symposium on Security and Privacy*, San Francisco, CA, 2013, pp. 80–94.

[11] J. Brozycki. (2008, Sep 13). "Detecting and preventing anonymous proxy usage," [Online]. Available: https://www.sans.org/reading-room/whitepapers/detection/detecting-preventing-anonymous-proxy-usage-32943

[12] T. Wang and I. Goldberg, "Improved website fingerprinting on Tor," *ACM Workshop on Privacy in the Electron. Soc.*, Berlin, Germany, 2013, pp. 201–212.

[13] P. Winter and S. Lindskog. (2012, Oct 10). How China is blocking Tor. CoRR. [Online]. pp. 1–7. Available: http://arxiv.org/abs/1204.0447

[14] T. Elahi, K. Bauer, M. AlSabah, R. Dingledine, and I. Goldberg, "Changing of the guards," *ACM Workshop on Privacy in the Electron. Soc.*, Raleigh, NC, 2012, pp. 43–53.

[15] D. M. Goldschlag, M. G. Reed, and P. F. Syverson, "Hiding routing information," *Workshop on Information Hiding,* Cambridge, UK, 1996, pp. 132–145.

[16] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The second-generation onion router," *USENIX Security Symposium*, San Diego, CA, vol. 13, 2004, p. 1–18.

[17] Replay attacks. (n.d.). Microsoft. [Online]. Available: https://msdn.microsoft.com/en-us/library/aa738652(v=vs.110).aspx. Accessed Oct 3, 2016.

[18] C. Schum, "Correctly implementing forward secrecy," GIAC (GCIH) Gold Certification, Sans Institute, 2014. Available: https://www.sans.org/reading-room/whitepapers/bestprac/correctly-implementing-secrecy-35842.

[19] M. Leech, M. Ganis, Y. Lee, R. Kuris, D. Koblas, L. Jones. (1996, Mar). RFC 1928: SOCKS protocol version 5 [Online]. Available: https://www.ietf.org/rfc/rfc1928.txt

[20] B. Greschbach, T. Pulls, L. M. Roberts, P. Winter, and N. Feamster. (2016, Oct 11). The effect of DNS on Tor's anonymity. [Online]. pp. 1–16. Available: https://arxiv.org/abs/1609.08187

[21] Freedom and privacy online. (n.d.) TorProject.org. [Online]. Available: https://media.torproject.org/image/official-images/tor-circumvention-brochure-001c.pdf. Accessed Oct 5, 2016

[22] S. Harris and F. Maymi. (2016) CISSP All-in-One Exam Guide, 7th Edition [eBook]. Available: http://techbus.safaribooksonline.com/book/certification/cissp/9780071849265

[23] D. Jayasekar. (2013, Dec). Deep dive Tor—introduction [Online]. Available: http://www.insiderattack.net/2013/12/deep-dive-into-tor-introduction_19.html.

[24] Improving Tor's anonymity by changing guard parameters (2014, Oct 16). Torproject.org. [Online]. Available: https://blog.torproject.org/blog/improving-tors-anonymity-changing-guard-parameters.

[25] S. Kowsalya, "website fingerprinting using rraffic analysis attacks," Graduate School of Computer and Information Sciences, University of Wisconsin-Madison, Madison, WI, 2014. Available: http://pages.cs.wisc.edu/~salinisk/642/website_fingerprinting_paper.pdf

[26] W. Odom. (2013). *CCNA Routing and Switching 200–120 Official Cert Guide Library.* [eBook]. Available: http://techbus.safaribooksonline.com/book/certification/ccna/9780134440897

[27] D. L. Prowse (2014). *CompTIA ® Security + SY0-401 Cert Guide , Deluxe Edition, Third Edition*[eBook]. Available: http://techbus.safaribooksonline.com/book/certification/securityplus/9780133836523

[28] I. A. Shoukat, A. Al-Dhelaan, M. Iftikhar. (2013) Realization of correlation between Round Trip Time (RTT) and hop counts in packet switched networks. *Life Science Journal* [Online]. *10*(4), pp. 569–576. Available: http://www.lifesciencesite.com/lsj/life1004/074_B02131life1004_569_576.pdf

[29] J. Barker, P. Hannay, and C. Bolan. (2010, Jul). Using Traffic Analysis to Identify Tor Usage—A Proposed Study. [Online]. Available: https://www.researchgate.net/publication/221199663_Using_Traffic_Analysis_to_Identify_Tor_Usage_-_A_Proposed_Study

[30] TCP/IP and TCPdump (2016). Sans Institute. [Online]. Available: https://www.sans.org/security-resources/tcpip.pdf

[31] K. Claffy, Y. Hyun, K. Keys, M. Fomenkov, and D. Krioukov, "Internet mapping: from art to science," in *Cybersecurity Applications & Technology Conference For Homeland Security*, Washington, DC, 2009, pp. 205–211.

[32] B. K. J. Cox and C. Gerg. (2004). *Managing Security with Snort & IDS Tools* [eBook]. Available: http://techbus.safaribooksonline.com/book/networking/intrusion-detection/0596006616

[33] B. A. Forouzan, *TCP/IP Protocol Suite*, 4th ed. New York, McGraw-Hill Education, 2009, pp 376–443.

[34] C. V Wright, S. E. Coull, F. Monrose, and C. Hill, "Traffic Morphing : an efficient defense against statistical traffic analysis," in *Network and Distributed Security Symposium*, San Diego, CA, 2009, pp. 375–382.

[35] H. M. Moghaddam, B. Li, M. Derakhshani, and I. Goldberg, "SkypeMorph: protocol obfuscation for Tor bridges," in *ACM Conference on Comput. Commun. Security,* Raleigh, NC, 2012, pp. 97–108.

[36] P. Winter, T. Pulls, and J. Fuss, "Scramblesuit: a polymorphic network protocol to circumvent censorship," in *ACM Workshop on Workshop on Privacy in the Electronic Society*, Berlin, Germany, 2013, pp. 213–224.

[37] K. P. Dyer, S. E. Coull, and T. Shrimpton, "Marionette: A programmable network traffic obfuscation system," in *Usenix Security Symposium,* Washington, DC, 2015, pp. 367–382.

[38] Q. Wang, G. T. K. Nguyen, and N. Borisov, “CensorSpoofer : Asymmetric communication using IP spoofing for censorship-resistant web browsing,” in *ACM Conference on Comput. Commun. Security,* Raleigh, NC, 2012, pp. 121–132.

[39] Z. Weinberg, J. Wang, V. Yegneswaran, L. Briesemeister, S. Cheung, F. Wang, and D. Boneh, “StegoTorus : A camouflage proxy for the Tor anonymity system,” in *ACM Conference on Comput. Commun. Security,* Raleigh, NC, 2012, pp. 109–120.

[40] D. Fifield, N. Hardison, J. Ellithorpe, E. Stark, R. Dingledine, P. Porras, and D. Boneh, “Evading censorship with browser-based proxies,” in *Privacy Enhancing Technologies Symposium,* Vigo, Spain, 2012, pp. 239–258.

[41] Asia overview (2010). OpenNet Initiative [Online]. Available: http://access.opennet.net/wp-content/uploads/2011/12/accesscontested-asia.pdf

[42] J. Karlin, D. Ellard, A. W. Jackson, C. E. Jones, G. Lauer, D. P. Mankins, and W. T. Strayer, “Decoy routing: toward unblockable Internet communication,” in *USENIX Workshop on Free Open Communications on the Internet*, San Francisco, CA, 2011, pp. 1–6.

[43] B. Cheswick, H. Burch, and S. Branigan, “Mapping and visualizing the Internet,” in *USENIX Annual Technology Conference,* San Diego, CA, 2000, pp. 1–12.

[44] D. T. Davis, private communication, Jan. 2017.

[45] J. M. Seng, “Behavior-based network management : a unique model-based approach to implementing cyber superiority,” unpublished.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California